GUIDA AL PERSONAL

VIC 20

I N D I C E

T A V O L E

I N T R O D U Z I O N E

Questo manuale e' la raccolta di scritti sul VIC 20 COMMODORE ricavati da riviste inglesi ed americane, da manuali e libri e da esperienze dirette.

Scopo principale non e' di insegnare la programmazione in Basic di questo computer (anche se e' presente un capitolo relativo ai comandi ed alle funzioni), ma di portare a conoscenza dell' utente tutte quelle informazioni che ne renderanno possibile qualsiasi tipo di applicazione.

Essenzialmente si divide in cinque parti che possono essere lette separatamente :
IL SISTEMA OPERATIVO
L'INTEGRATO 6502 E LE RELATIVE FUNZIONI
L'INTEGRATO 6561
L'INGRATO 6522
LE TAVOLE

Ad alcuni di questi capitoli sono stati aggiunti dei piccoli programmi dimostrativi.
La EVM ha controllato questo scritto, tuttavia puo' essere sfuggito qualcosa sia per errori tipografici che per altri motivi.

Per questo motivo in fondo al volume e' presente una scheda in due esemplari per segnalare errori o per dare suggerimenti.
Dato che si tratta di copie numerate si invita gli utenti di questo manuale a scriverci: le loro lettere saranno prese in considerazione, i loro nominativi inseriti in un MAILING-LIST e ove vengano riscontrati errori, a tutti verra' inviata la pagina sostitutiva.

Ci auguriamo che questa collaborazione possa portare a risultati proficui per tutti.

M A P P A D I M E M O R I A D E L V I C

Il microprocessore 6502 usato nel VIC e' capace di indirizzare fino a 64 Kbytes di memoria. Questo spazio di memoria e' diviso in blocchi ognuno dei quali ha delle specifiche funzioni secondo lo schema dato dal Sistema Operativo.
Nella configurazione base del VIC , alla quale ci riferiamo costantemente, solo 29 k dei 64 k disponibili e' utilizzata Gli altri 35 Kbytes sono disponibili per le espansioni dove e' possibile utilizzare memorie ROM o RAM o I/O specializzati, tutti accessibili tramite il connettore di espansione posto sul retro della macchina e del quale parleremo in seguito.
La suddivisione dello spazio di memoria in blocchi con le loro differenti funzioni ed indirizzi e' schematizzata nella tavola 1,mentre in appendice, si trova una mappa di memoria piu' completa.
La comprensione delle funzioni ed il collocamento di ogni blocco e' essenziale per poter utilizzare in pieno la potenza di questo elaboratore.

1 - VARIABILI DI SISTEMA - indirizzi esadecimali da esa 0000 a esa 03ff - decimale da 0 a 1023-

I primi 1023 bytes di memoria RAM sono utilizzati dal BASIC e dal SISTEMA OPERATIVO per l' immagazzinamento (STORAGE) delle variabili di sistema.
La configurazione del SISTEMA VIC ed il suo modo di operare possono ,in altre parole, essere cambiati variando i valori esistenti entro le locazioni di questa parte di memoria.

2 - MEMORIA RAM UTENTE - esa da esa 0400 a esa 7fff - decimale da 1024 a 32767

ESAD.		DEC.
0000 03FF	1K RAM per area di lavoro S.O e BASIC	0000 1023
0400 1DFF	area per espansione da 3K	1024 7679
1E00 1FFF	area per RAM schermo	7680 8191
2000 3FFF	8K espansione RAM/ROM	8192 16383
4000 5FFF	8K espansione RAM/ROM	16384 24575
6000 7FFF	8K espansione RAM/ROM	24576 32767
8000 8FFF	area generatore di caratteri 4K ROM	32768 36863
9000	registri 6561	36864
9110	registri 6522 1 e 2	37136
9600	area RAM del Colore	38400
9800	espansione I/O Blk. 2	38912
9C00	espansione I/O Blk. 3	39936
A000 BFFF	8K espansione ROM	40960 49151
C000 DFFF	8K ROM BASIC	49152 57343
E000 FFFF	8K ROM per S.O	57344 65535

MAPPA DI MEMORIA DEL VIC - 20

Questi 31 Kbytes di memoria utente possono a loro volta essere suddivisi in 4 sottoblocchi di cui il primo di 7k di lunghezza e gli altri 3 di 8k.
Il primo sottoblocco consiste esclusivamente di memoria RAM ed e' formato da 4k di memoria utente che va da esa 1000 a esa 1fff, piu' 3k da esa 0400 a esa 0fff

Se non ci sono altre espansioni RAM allora i 512 Bytes piu' alti della prima sezione RAM (da esa 1e00 a esa 1fff) sono usati come memoria di schermo.

Se e' usato l' HIGH RESOLUTION MODE in aggiunta alla memoria di schermo sono allora necessari 4k di RAM per il generatore di caratteri programmabile come spiegheremo nel capitolo dedicato al 6561.

Inoltre se sono presenti piu' di 7k RAM di memoria utente il sistema si riconfigura automaticamente e sposta la memoria video a partire dalla locazione esadecimale esa 1000 (decimale 4096).

Gli altri 3 sottoblocchi di memoria utente ciascuno da 8k possono essere composti sia da ROM che da RAM e sono completamente liberi per programmi e dati.

3 -- GENERATORE DI CARATTERI -- esa da esa 8000 a esa 8fff - decimale da 32768 a 36863

Il generatore di caratteri e' una ROM da 4 Kbytes che contiene un insieme di matrici usate per far comparire ognuno dei 255 caratteri del VIC sullo schermo del monitor.

Il contenuto del generatore di caratteri dipende dal tipo di VIC che si possiede.
Attualmente ne esistono 2 versioni : quella giapponese con i caratteri KATAKANA tipici di quella lingua ed una standard per europei ed americani. La ROM puo' essere sostituita da una

EPROM appositamente riprogrammata con altri tipi di caratteri.

Normalmente pero' all' utente non e' necessario cambiare la disposizione dei caratteri ne' il modo in cui gli stessi sono immagazzinati.
Tuttavia nell' HIGH RESOLUTION DISPLAY MODE il generatore di caratteri non e' usato per cui per avere il display dei caratteri alfanumerici standard del VIC sara' necessario caricare nella parte RAM del GENERATORE DI CARATTERI PROGRAMMABILE il contenuto di tutto o di parte del GENERATORE DI CARATTERI STANDARD.

4 - INPUT OUTPUT E CONTROLLO INTERFACCE - esa da esa 9000 a 912f decimale da 36864 a 37176.

Questa parte di memoria contiene la mappa di tutti gli indirizzi del sistema di INPUT. OUTPUT e CONTROLLO DELLE LINEE.
Cio' consente che una linea di I/O possa essere utilizzata (messa cioe' in ON o in OFF) cambiando il corrispondente BIT di quella specifica locazione di memoria.
Infatti i registri interni dei 3 CHIPS di I/O sono accessibili attraverso questa parte di memoria.
I 3 CHIPS sono due 6522 VIA ed il 6521 VIC che controlla le operazioni di video display.
La comprensione del funzionamento di questi integrati e' essenziale per l'uso di una qualsiasi delle numerose interfacce del VIC e per il collegamento con periferiche esterne e questa e' la ragione per cui gran parte di questo manuale sara' dedicata all' uso di questi registri e delle periferiche ad esso collegabili.

5 - MEMORIA DEL COLORE - esa da esa 9600 a esa 97ff - decimale da 38400 a 38911.

Ognuno dei 506 BYTES presenti in questo blocco di memoria determina il colore dello schermo o del

fondo.
E' importante notare che se la memoria utente e'maggiore di 7K di RAM anche in questo caso il sistema si riconfigura automaticamente spostando l' inizio della memoria colore da esa 9600 a esa 9400 (decimale 37888

6 - ESPANSIONE DI MEMORIA ROM - esa da esa a000 a esa bfff - decimale da 40960 a 49151.

Questo blocco da 8k di memoria e' riservato per l' uso di programmi immagazzinati in ROM ed allocati in un qualsiasi cartridge di ROM,RAM o EPROM inseriti nella porta di espansione del VIC.
Il sistema operativo del VIC consente che un programma in linguaggio macchina, che abbia inizio dalla locazione esa a000, parta direttamente all' accensione della macchina (vedi il capitolo INIZIALIZZAZIONE DEL SISTEMA e AUTOPOWER UP).

7 - INTERPRETE BASIC DEL VIC - esa da esa c000 a esa dfff - decimale da 49152 a 57343.
L' Interprete ha la funzione di tradurre un programma scritto in linguaggio ad alto livello, appunto il BASIC, in una serie di subroutines in linguaggio macchina direttamente eseguibili passo passo dal calcolatore.

8 - SISTEMA OPERATIVO DEL VIC - esa da esa e000 a esa ffff - decimale da 57344 a 65535.

Il sistema operativo ha essenzialmente la funzione di controllare interamente il funzionamento del VIC.
Controlla cioe' le comunicazioni fra l'interno e l'esterno, il funzionamento del Basic, inizializza il sistema all' accensione, controlla lo schermo sia come editing che come suoni e colore ,ecc.
Il sistema operativo funziona di norma insieme all' interprete BASIC ma le sue routines possono essere adoperate per facilitare la scrittura di programmi in linguaggio macchina.

E S P A N S I O N E   D E L L A   M E M O R I A

Si possono aggiungere al VIC sia delle linee di Input/Output sia della memoria tramite il connettore di espansione della memoria.
Si tratta di usare connettori a 44 pin con spazi di 0.156 pollici il cui uso e' riportato nella tavola 2.
Si ricorda che e' necessaria una grande cura per l' uso di questi connettori, sia per l' inserimento in maniera tale da non provocare danni fisici, sia per l' eventuale uso di piastre autocostruite.
Infatti queste linee non sono Bufferizzate per cui ogni malfunzionamento puo' provocare danni al Sistema.
Le linee della porta di espansione si possono dividere in 5 grandi gruppi:
BUS DATI -- otto linee usate per trasferire dati tra il processore e la memoria.

BUS DI INDIRIZZI - consiste in 14 linee (le meno significative) di indirizzi.

BUS DI CONTROLLO - consiste in 6 linee di controllo del sistema : IRQ, CLOCK, RESET, READ/WRITE, NMI

SELEZIONE BLOCCHI DI INDIRIZZO - ci sono 9 blocchi di selezione, che sono generati da una parziale decodifica degli indirizzi piu' significativi.
Questi sono usati per selezionare i blocchi di memoria o gli indirizzi di ingresso/uscita del bus I/O.

LINEE DI ALIMENTAZIONE - sono disponibili la massa e i + 5 Volts con circa 750 ma.

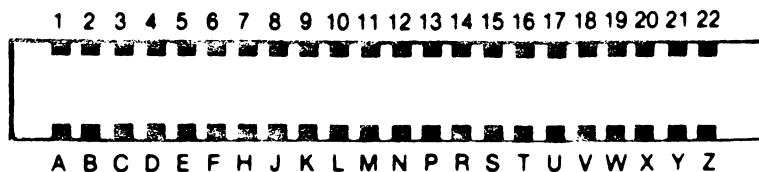
I segnali disponibili nell' espansione di memoria sono come nella tabella sottoriportata.

NOME	PIN	DESCRIZIONE
GND	1	MASSA DEL SISTEMA
CD0	2	BUS DATI LINEA 0
CD1	3	BUS DATI LINEA 1
CD2	4	BUS DATI LINEA 2
CD3	5	BUS DATI LINEA 3
CD4	6	BUS DATI LINEA 4
CD5	7	BUS DATI LINEA 5
CD6	8	BUS DATI LINEA 6
CD7	9	BUS DATI LINEA 7
BLK1	10	SELEZIONE DEL I BLOCCO DA 8K CON START A esa 2000
BLK2	11	SELEZIONE DEL II BLOCCO DA 8K CON START A esa 4000
BLK3	12	SELEZIONE DEL III BLOCCO DA 8K CON START A esa 6000
BLK5	13	SELEZIONE DEL V BLOCCO DA 8K CON START A esa A000
RAM1	14	SELEZIONE DEL I BLOCCO DA 1K CON START A esa 0400
RAM2	15	SELEZIONE DEL II BLOCCO DA 1K CON START A esa 0800
RAM3	16	SELEZIONE DEL III BLOCCO DA 1K CON START A esa 0C00
VR/W	17	LINEA DI READ/WRITE BUFFERIZZATA
CR/W	18	LINEA DI READ/WRITE DALLA CPU
IRQ	19	LINEA DI IRQ DEL 6502
NC	20	NON CONNESSO
+5V	21	LINEA DI ALIMENTAZIONE A 5 VOLTS
GND	22	MASSA DEL SISTEMA
GND	A	MASSA DEL SISTEMA
CA0	B	LINEA 0 DELL' INDIRIZZO
CA1	C	LINEA 1 DELL' INDIRIZZO
CA2	D	LINEA 2 DELL' INDIRIZZO
CA3	E	LINEA 3 DELL' INDIRIZZO
CA4	F	LINEA 4 DELL' INDIRIZZO
CA5	H	LINEA 5 DELL' INDIRIZZO
CA6	J	LINEA 6 DELL' INDIRIZZO
CA7	K	LINEA 7 DELL' INDIRIZZO
CA8	L	LINEA 8 DELL' INDIRIZZO

CA9 M LINEA 9 DELL' INDIRIZZO
CA10 N LINEA 10 DELL' INDIRIZZO
CA11 P LINEA 11 DELL' INDIRIZZO
CA12 R LINEA 12 DELL' INDIRIZZO
CA13 S LINEA 13 DELL' INDIRIZZO
I/O2 T SELEZIONE DEL II BLOCCO DI I/O START A esa 9130
I/O3 U SELEZIONE DEL III BLOCCO DI I/O START A esa 9140
SO2 V CLOCK 2 DEL SISTEMA
NMI W LINEA DI NMI DEL 6502
RESET X LINEA DI RESET DEL 6502
NC Y NON CONNESSA
GND Z MASSA DEL SISTEMA

MEMORY EXPANSION

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

A B C D E F H J K L M N P R S T U V W X Y Z

PIN #	TYPE
1	GND
2	CDØ
3	CD1
4	CD2
5	CD3
6	CD4
7	CD5
8	CD6
9	CD7
1Ø	$\overline{\text{BLK1}}$
11	$\overline{\text{BLK2}}$

PIN #	TYPE
12	$\overline{\text{BLK3}}$
13	$\overline{\text{BLK5}}$
14	$\overline{\text{RAM1}}$
15	$\overline{\text{RAM2}}$
16	$\overline{\text{RAM3}}$
17	VR/W
18	CR/W
19	$\overline{\text{IRQ}}$
2Ø	NC
21	+5V
22	GND

PIN #	TYPE
A	GND
B	CAØ
C	CA1
D	CA2
E	CA3
F	CA4
H	CA5
J	CA6
K	CA7
L	CA8
M	CA9

PIN #	TYPE
N	CA1Ø
P	CA11
R	CA12
S	CA13
T	I/Ø2
U	I/Ø3
V	SØ2
W	$\overline{\text{NMI}}$
X	$\overline{\text{RESET}}$
Y	NC
Z	GND

PIEDINATURA SUL CONNETTORE DI ESPANSIONE

L E   V A R I A B I L I   D I   S I S T E M A

L' intero blocco di memoria che va dalla locazione decimale 0 alla 1023 e' riservata dal sistema operativo ed e' usata per contenere le variabili di sistema, per l'immagazzinamento temporaneo di dati e per i BUFFER di I/O.
Questa area di memoria e' accessibile all' utente attraverso i comandi BASIC PEEK e POKE e anche caricando programmi in linguaggio macchina.
ATTENZIONE
le locazioni che vanno dal 256 al 511 contengono lo STACK PROCESSOR e non dovrebbero essere utilizzate se non tramite esclusivo di programmi in linguaggio macchina e con notevole cautela.

La possibilita' di accedere a questa parte di memoria e' di enorme importanza in quanto numerose funzioni non possono essere implementate senza una perfetta conoscenza del funzionamento di queste locazioni.
Questa area di memoria i cui indirizzi e le cui funzioni specifiche sono riportate in fondo al capitolo, puo' a grandi linee essere divisa in 7 grandi sottoblocchi.

1 - VARIABILI DELL' INTERPRETE BASIC - esa da esa 0000 a esa 008f - decimale da 0 a 143.
Questa sezione della pagina zero e' usata esclusivamente dall' interprete BASIC.Dalla locazione decimale 43 alla 74 sono presenti i 16 piu' importanti puntatori ognuno dei quali occupa 2 bytes di cui uno per la parte alta dell'indirizzo e l' altro per la parte bassa.
Se in un programma BASIC sono usate subroutines in linguaggio macchina, allora le locazioni da 87 a 96 possono essere usate per l' immagazzinamento di variabili in pagina zero.
Se sta girando un programma in linguaggio macchina che non richieda l' utilizzo dell' interprete BASIC allora l' intera zona di memoria (da 0 a 143) puo' essere usata per l' immagazzinamento di variabili.

Se viene usato un comando USR oppure se una qualsiasi routine di funzionamento del BASIC e' chiamata attraverso un programma in linguaggio macchina allora il FLOATING ACCUMULATOR da 97 a 102 sara' utilizzato per il trasferimento delle variabili.

2 - VARIABILI DEL SISTEMA OPERATIVO (1) - esa da esa 90 a esa ff - decimale da 144 a 255.

Tutte le variabili ed i parametri immaggazzinati in questa ultima parte della pagina zero sono di grande interesse per il programmatore.
Queste infatti controllano le funzioni di input e output del VIC relativamente alle porte RS232 e IEEE-488, l' allocazione dei FILES per le operazioni di I/O relativamente alle periferiche, il trasferimento di dati fra il VIC e l' unita' a cassette, il controllo dello schermo ecc.

3 - STACK PROCESSOR - esa da esa 0100 a esa 01ff - decimale da 256 a 511

La sezione di memoria occupata dallo stack e' comune a tutti i calcolatori che sono forniti del processore 6502.
Le operazioni sullo STACK e' automaticamente controllata dal processore e qualsiasi accesso a questa area di memoria deve essere effettuato con grande cautela.
per questo si consiglia di rileggere accuratamente la parte dedicata al microprocessore 6502 ed al linguaggio macchina.

4 - BUFFER DEL BASIC - esa da esa 0200 a esa 0258 - decimale da 512 a 600.

Questi 88 bytes sono usati per immagazzinare temporaneamente testi o linee di programma.
RICORDARSI che una linea di programma puo' essere lunga al massimo 88 caratteri.
Quando viene inserita una linea di programma questa viene trasferita dal BUFFER DEL BASIC alla memoria con un RETURN (o ritorno carrello).
La linea di programma e' allora convertita nella forma canonica di immagazzinamento e come vedremo i comandi vengono interpretati e ridotti a TOKENS (dati) per un 'ottimizzazione degli spazi.
Anche nell'accesso a questa area di memoria e' necessaria un' estrema cautela.

5 - VARIABILI DEL SISTEMA OPERATIVO (2) - esa da esa 0259 a esa 02ff decimale da 601 a 767.

Questi parametri del sisstema operativo e le relative variabili . non richiedono di essere

allocati in pagina zero e sono praticamente la continuazione di quelle viste nel paragrafo 2.
Tutte le locazioni di questa sezione possono essere accessibili senza particolari cautele.

6 -- INDIRIZZI INDIRETTI E VETORI DI SALTO - esa da esa 0300 a esa 0334 -- decimale da 768 a 820.

Questa parte di memoria e' utilizzata per immagazzinare indirizzi di salto indiretti per le funzioni di sistema ed e' di notevole interesse.
Alcuni degli indirizzi indiretti sono immagazzinati temporaneamente qui e non sono di grande importanza, mentre 16 indirizzi sono immagazzinati in maniera permanente e sono importantissimi perche' controllano la maggior parte del sistema operativo.

Questi indirizzi possono essere usati per accedere alle routines del sistema operativo, per intercettarle, per modificarle oppure anche per rimpiazzarle interamente con nuove funzioni.
Per esempio inserendo determinati codici dentro le routine di scansione della tastiera e modificando il temporizzatore CLOCK si puo' ottenere di controllare automaticamente gli input della user port.
Si possono anche cambiare completamente le routines del sistema operativo
Ad esempio se sono collegate delle periferiche non standard al VIC allora sara' indispensabile variare le routines di INPUT e OUTPUT.
L' uso di questa parte della memoria di cui si raccomanda un accurato studio, oltre a dare una grande flessibilita' al sistema consete di ridefinire i tutto o in parte il sistema operativo per adattarlo a particolari situazioni.

7 - BUFFER DELLA CASSETTA - esa da esa 0334 a esa 03fc - decimale da 821 a 1020.

Questa area di buffer di 198 Bytes e' usata durante il trasferimento di dati fra il VIC e la cassetta. In particolare vi si controlla l' immagazzinamento di ogni blocco di dati.
Se non e' usata la cassetta questa parte di memoria puo' liberamente essere utilizzata per immaggazinare piccoli programmi in linguaggio macchina.

M A P P A   D I   M E M O R I A

==== I N D I R I Z Z I ====		F U N Z I O N E
ESA	DECIMALE	
0000-0002	0-2	funzione USR
0003-0004	3-4	converte floating in intero
0005-0006	5-6	converte intero in floating
0007	7	contatore generale del BASIC. Ricerca il carattere ":" o la fine della linea
0008	8	delimitatore di modo apici,00 come limite
0009	9	posizione del cursore
000a	10	flag di verifica
000b	11	puntatore del buffer di input
000c	12	DIM flag.Primo carattere della matrice
000d	13	Variable flag. ff=stringa 00=numerico
000e	14	Integer flag. 80= integer 00=float. point
000f	15	Flag scansione DATA.LIST Flag.Memory flag
0010	16	Subscript flag.FNx flag
0011	17	Flags per :0=input;64=get;1=read
0012	18	Flag per segno ATN
0013	19	attuale canale di I/
0014-0015	20-21	Indirizzo basic per SYS,GOTO,ecc
0016	22	Stack Pointer per temporary string
0017-0018	23-24	Ultimo vettore per temporary string
0019-0021	25-33	Stack per stringhe temporanee
0022-0025	34-37	area di utilizzo per punt.
0026-002a	38-42	area di utilizzo per molt.
002b-002c	43-44	puntatore inizio basic
002d-002e	45-46	punt.fine programma e inizio variabili
002f-0030	47-48	punt.fine variabili inizio array
0031-0032	49-50	punt.fine array
0033-0034	51-52	punt.inizio zona stringhe
0035-0036	53-54	punt.fine zona stringhe
0037-0038	55-56	punt.fine memoria
0039-003a	57-58	n. della attuale linea Basic
003b-003c	59-60	successiva linea Basic
003d-003e	61-62	puntatore per com.Basic (CONT)
003f-0040	63-64	n. della linea DATA
0041-0042	65-66	punt. per il DATA in esame

0043-0044	67-68	vettore di input
0045-0046	69-70	nome della var. corrente
0047-0048	71-72	indirizzo della var. corrente
0049-004a	73-74	puntatore per FOR-NEXT
004b-004c	75-76	salva Y,salva oper.,salva punt. basic
004d	77	maschera per l'operatore in uso
004e-0053	78-83	ara di utilizzo : punt. alla descrizione della stringa, lunghezza della stringa,etc.
0054-0056	84-86	vettore di salto per funzioni
0057-0060	87-96	area di utilizzo per valori num.
0061-0066	97-102	Accumulatore £1: E,M,M,M,M,S
0068	104	Accumulatore £1: Overflow
0069-006e	105-110	Accumulatore £2: Esponente,etc
006f	111	comparazione segno
0070	112	Accumulatore £1: basso ordine
0071-0072	113-114	lungh. buffer cassetta
0073-008a	115-138	Subrtn: Get;7A.7B = puntatore (CHARGOT)
008b-008f	139-143	area di memorizzazione per RND
0090	144	flag per le operazioni di I/o
0091	145	flag per il tasto di stop
0092	146	temporizzatore
0093	147	flag per LOAD o VERIFY
0094	148	uscita seriale, flag caratt riman.
0095	149	caratt.bufferizzato per IEEE
0095	150	sincronizz. cassetta
0097	151	temporizz. per INPUT sulla IEEE
0098	152	n. di files aperti
0099	153	sgn. dellla perif.in INPUT
009a	154	periferica in OUTPUT per CMD
009b	155	controllo di parita' per cass
009c	156	dipolo per cassetta
009d	157	flag per mess. da S.O per RUN o per moo dir.
009e	158	tempor. cassetta. Errore 1
009f	159	" " Errore 2
00a0-00a2	160-162	Jiffy clock
00a3	163	Contatore seriale per bit
00a4	164	Cont. di ciclo per I/O seriale
00a5	165	cont. per scrittura nastro
00a6	166	puntatore per Buffer di cass.

00a7-00b6	167-182	area di lavoro per RS-232
00b7	183	numero dei carat. nel nome del file
00b8	184	numero del file logico corrente
00b9	185	indirizzo secondario corrente
00ba	186	numero della periferica corrente
00bb-00bc	187-188	puntatore al nome del file
00be	190	£ resto del blocco per R/W
00bf	191	buffer della parola seriale
00c0	192	controllo switch cassetta
00c1-00c2	193-194	indirizzo di start per cass.
00c3-00c4	195-196	puntatore kernal
00c5	197	valore del tasto premuto
00c6	198	numero dei caratteri nel buffer
00c7	199	flag di reverse (0 off)
00c8	200	puntatore di fine linea per input
00c9-00ca	201-202	posiz. cursore (riga e colonna)
00cb	203	stesso del 197
00cc	204	lampeggio cursore (0=on 1=off)
00cd	205	flashing cursore
00ce	206	carattere prima del cursore
00cf	207	flag lampeggio del cursore
00d0	208	input da schermo e da tastiera
00d1-00d2	209-210	puntatore alla linea dello schermo
00d3	211	posizione del cursore nella linea
00d4	212	flag delle " (0=off 1=on)
00d5	213	lunghezza delle linee sullo schermo
00d6	214	riga dello sch. in cui si trova il cur.
00d7	215	valore ascii dell'ultimo tasto prem.
00d8	216	flag in modo INSERT
00d9-00f1	217-241	tavola delle linee dello schermo
00f2	242	riga segnata sullo,schermo
00f3-00f4	243-244	puntatore del colore dello schermo
00f5-00f6	245-246	puntatore tastiera
00f7-00f8	247-248	RS-232 puntatore di ricezione
00f9 00fa	249-250	RS-232 puntatore di trasmissione
00fb-00ff	251-255	loc. in pag. zero delle rout. kernal
0100-010a	256-266	area di lavoro ascii
010b-013e	267-318	area di errore nastro
013f-01ff	319-511	area stack del processore
0200-0258	512-600	buffer della riga basic
0259-0262	601-610	tavola del file logico

0263-026c	611-620	tavola del num. della perif.
026d-0276	621-630	tavola dell'indirizzo secondario
0277-0280	631-640	buffer della tastiera
0281-0282	641-642	punt. dell'inizio della memoria
0283-0284	643-644	punt. della fine memoria
0285	645	flag del fuori tempo per IEEE
0286	646	codice del colore corrente
0287	647	codice del colore prima del cursore
0288	648	loc.di base dello schermo (MSB)
0289	649	massima dimensione del buffer tastiera
028a	650	flag di repeat(0=solo cursore, 128=tutt
028b	651	ritardo prima del repeat
028c	652	ritardo tra i repeat
028d	653	flag shift/control tastiera (shift=1, commodore = 2 , ctrl = 4)
028f-0290	655-656	punt. decod. della tastiera
0291	657	loc. modo shift
0292	658	auto scroll basso (0=on 0=off)
0293	659	RS-232 registro di controllo
0294	660	RS-232 registro di comando
0295-0296	661-662	non standard
0297	663	RS-232 registro di stato
0298	664	numero dei bit da inviare
0299-029a	665-666	baud rate (full) bit time
029b	667	RS-232 punt. di ricez.
029c	668	RS-232 punt. di input
029d	669	RS-232 punt. di trasm.
029e	670	RS-232 punt. di outp.
029f-02a0	671-672	mantiene IRQ durante oper. su nastro
02a1-02ff	673-767	liberi
0300-0301	768-769	vettore di salto per errore
0302-0313	770-787	vettori di salto per basic
0314-033b	788-827	vettori di salto rout. kernal
033c-03fb	828-1019	buffer della cassetta

V I C U S E R M E M O R Y

L' ammontare di memoria disponibile per l' utente dipende da quanta memoria RAM viene aggiunta tramite i cartriges di espansione ed e' comunque compresa fra un minimo di 3 Kbytes della configurazione di base ed un massimo di 31 Kbytes nella sua massima attuale espansione.
Come accennato questa memoria non e' tuttavia completamente disponibile per immagazzinare programmi in quanto una parte viene riservata per la memoria di schermo e per immagazzinare stringhe e variabili numeriche.

Infatti scrivendo un programma di 3K di lunghezza in un VIC in configurazione base e facendolo girare si otterrebbe un :
OUT OF MEMORY ERROR
perche' il sistema operativo si accorgerebbe della mancanza di spazi dedicati alle variabili che in un programma devono pur essere presenti.

Nella configurazione base del VIC i programmi sono immagazzinati a partire dalla locazione di memoria decimale 4097 in poi mentre le stringhe e le variabili sono immagazzinate dalla piu' alta locazione di memoria disponibile contando pero' all' indietro.
Stesso concetto per eventuali espansioni con la differenza che l' indirizzo di partenza non e' 4097 ma e' spostato in rapporto al tipo di espansione connessa

IMMAGAZZINAMENTO E COMPATTAZIONE DEI PROGRAMMI

Come abbiamo visto quando si inserisce una linea di programma tramite tastiera il VIC per prima cosa la scrive nel BUFFER di TASTIERA Il sistema operativo non esegue nessuna trasformazione ma trascrive semplicemente la linea da come appare sullo schermo.
La linea di programma tuttavia non viene memorizzata fino a quando non si preme il RETURN.
L' operazione di ritorno carrello consente al sistema operativo di trasferire la linea appena inserita sullo schermo dentro la memoria RAM attraverso il buffer del basic dove la linea di programma e' compressa e formattata.
Ogni linea e' immagazzinata in uno specifico formato che ne riduce l'occupazione di memoria e consente quindi di scrivere programmi piu' lunghi.
Ogni comando BASIC viene ad esempio convertito in un singolo BYTE per cui il comando RESTORE invece di occupare 7 BYTES sara' immagazzinato come un singolo BYTE del valore decimale di 140.
Quando invece il programma e' LISTATO viene eseguita l'operazione inversa ritrasformando il valore 140 nel comando RESTORE e rendendone agevole la visualizzazione.

Un vantaggio accessorio ma non irrilevante del procedimento di compressione dei testi e' dato dal sistema di scrivere i comandi BASIC sia nel programma che in modo diretto.
Occorre tenere presente infatti che la routine che converte i comandi in valori decimali non esamina tutta la parola ma solo i primi due o tre caratteri dello STATEMENT.
Di norma pero' immettendo solo i primi due caratteri del comando il computer rispondera' con un messaggio di errore dovuto all' intervento della ERROR DETECTION ROUTINE. Per superare questa

routine sara' sufficiente scrivere la prima lettera del comando in maniera normale e la seconda usando anche il tasto SHIFT.
Nei casi in cui possa esserci confusione perche' due comandi hanno la stessa coppia di iniziali si digitera' le prime due lettere normalmente e la terza con lo SHIFT.
La tavola 3 riporta l'elenco dei comandi Basic, la descrizione abbreviata ed il codice decimale con il quale viene immaggazzinato.
La tavola delle RESERVED COMMAND WORDS inizia all' indirizzo 49310 fino a 49566.
Un programma BASIC e' immagazzinato come una serie di blocchi di lunghezza variabile ognuno dei quali rappresenta una linea di programma.
Ogni blocco ha un formato fisso e tutti insieme sono riuniti ed interconnessi tramite un struttura di concatenamento (LINK).

L' esempio sottoriportato servira' a chiarire meglio di ogni altro discorso il sistema di immagazinamento dei programmi. L' esempio si riferisce ad una configurazione standard del VIC e quindi con partenza da 4096 e si riportano le prime due linee di un qualsiasi programma.

```
10input a
20if int (a)  5 then print aß2
30.........................
```

indirizzo decimale	contenuto decimale	video	commenti
4096	0		Il primo byte e' 0
4097	8		Indirizzo di link parte bassa
4098	4		indirizzo di link parte alta
4099	10	10	numero di linea parte bassa
4100	0		numero di linea parte alta
4101	133	input	comando
4102	65	A	variabile A
4103	0		separatore di linea
4104	29		link bassa
4105	4		link alta
4106	20		numero di linea bassa
4107	0		numero di linea alta
4108	139		comando IF
4109	32		spazio fra i comandi
4110	181		comando INT
4111	32		spazio fra i comandi
4112	40	(	parentesi
4113	65	A	variabile
4114	41	)	parentesi
4115	32		spazio

NOTA: I numeri di linea sono immagazzinati in due bytes per cui il nume
piu' alto che si potrebbe ottenere e' 65535, in pratica pero' si riduce
63999

Quando un programma sta girando il numero di linea in esecuzione e' pos
nelle locazioni 57 e 58 della pagina zero.

L' indirizzo di link e' invece caricato nelle locazioni 122 e 123.

I L S I S T E M A O P E R A T I V O D E L VIC

Come abbiamo detto in precedenza i 16K Bytes piu' alti della memoria sono occupati dal Sistema Operativo.
Il Sistema operativo e' costituito da una serie di programmi scritti in linguaggio macchina che consentono al VIC di far funzionare l' interprete Basic, di comunicare con le periferiche, oltre alle operazioni di input ed output attraverso il video e la tastiera.
Questi 16k di S.O. sono importantissimi in quanto consentono al VIC di funzionare come sistema, cioe' di funzionare in un certo modo.
Infatti l' hardware del VIC e' cosi' flessibile che cambiando una parte del software del Sistema Operativo e' possibile ridefinire completamente la macchina e quindi il suo funzionamento.
Ad esempio per quanto riguarda le periferiche in uso, e' possibile ridefinire via software, cioe' cambiando alcuni parametri del S.O., l' uscita RS232 per adattarla allo standard CENTRONIC.
Con lo stesso sistema e' possibile far girare sul VIC altri linguaggi ad alto livello sostituendoli semplicemente all' interprete BASIC, purche' ovviamente questi nuovi linguaggi siano compatibili con il S.O.
I 16 K Bytes possono essere suddivisi in due grandi parti:

AREA INTERPETRE BASIC

AREA OPERATING SISTEM

Ognuna di queste due grandi aree e' approssimativamente della stessa ampiezza, cioe' di circa 8K Bytes.
L' interprete Basic va dalle locazioni esadecimali $c000 a $dfff, mentre il Sistema Operativo occupa

le posizioni ROM da $e000 a $ffff cioe' fino alla fine della memoria.
Il sistema operativo e' un gruppo di programmi a se stante per cui non necessita degli 8 kBytes dell' interprete per funzionare mentre l' Interprete Basic usa numerose routine del S.O. per funzionare.
In particolare le Routines per l' I/O e le funzioni di comunicazione con le periferiche del Sistema Operativo sono usate dall' Interprete.
Sia il Basic che il Sistema Operativo adoperano la parte piu' alta disponibile della memoria RAM e la pagina ZERO per trasferire fra di loro variabili e per i puntatori.
Nel Basic la maggior parte dei calcoli avviene con l' uso dei numeri in Floating Point invece dei semplici Integers o dei valori binari.
Di conseguenza la maggior parte di queste routines che rendono possibili queste funzioni utilizzano i FLOATING POINT ACCUMULATORS allocati in pagina zero.
Riportiamo in tabella il formato e le locazioni degli accumulatori:

Acc. n. 1	Acc. n. 2	Funzione
$61	$69	Esponente + $80
$62	$6a	Frazione MBS in binario
$63	$6b	Frazione del Byte 2
$64	$6c	Frazione del Byte 3
$65	$6d	Frazione LSB
$66	$6e	Segno(ff= -; 00= +)
$6f		Byte di confronto segno
$70		Byte flutt. per Acc. 1

La maggior parte delle Routines usate sia dall' Interprete Basic che dal Sistema Operativo possono essere richiamate ed usate da parte di programmi scritti in Linguaggio Macchina su altri elaboratori.
Per quanto detto all' inzio di questo capitolo e cioe' per la grande flessibilta' che i cambiamenti di questo Software o semplicemente il loro particolare uso possono consentire nell' uso del VIC, ci soffermeremo a lungo su questo argomento trattando:

L ' ELENCO DI QUESTE ROUTINES

LA LOCALIZZAZIONE

IL SIGNIFICATO

UN APPROFONDIMENTO PER LE PIU' IMPORTANTI

Ricordiamo per inciso che il Basic del VIC e' stato scritto dalla MICROSOFT ma ampiamente modificato in seconda stesura dalla stessa COMMODORE.

T A V O L E   D E L   S I S T E M A   O P E R A T I V O

C000-C045	INDIRIZZO DI AZIONE PER TASTIERA
C046-C073	INDIRIZZI PER LE FUNZIONI
C074-C091	INDIRIZZI PER GLI OPERARTORI
C092-C192	PAROLE CHIAVI DEL BASIC
C193-C2A9	MESSAGGI DI ERRORE
C38A-C3B7	RICERCA DELLO STACK PER FOR E GOSUB
C3B8-C3FA	APERTURA SPAZIO IN MEMORIA
C3FB-C407	TEST PER LO STACK
C408-C434	CONTROLLO DELLA MEMORIA DISPONIBILE
C435	MANDA UN MESSAGGIO DI ERRORE
C474-C482	STAMPA READY
C483-C532	MANOVRA DI UN NUOVA LINEA DA TASTIERA
C533-C55F	RICOSTRUZIONE CONCATENATA DI UNA LINEA BASIC
C560-C57B	RICEVE LINEA DALLA TASTIERA
C57C-C612	COMPATTAZIONE COMANDI BASIC
C613-C641	RICERCA UN NUMERO DI LINEA BASIC DATO
C642	ESEGUEIL COMANDO NEW
C660-C68D	ESEGUE CLR
C68E-C69B	RESET DEL BASIC ALLA PARTENZA
C69C-C741	ESEGUE LIST
C742-C7EC	ESEGUE FOR
C7ED-C81G	ESEGUE UN COMANDO BASIC
C81D-C82B	ESEGUE RESTORE
C82C-C856	ESEGUE STOP E END
C857-C870	ESEGUE CONT
C871-C882	ESEGUE RUN
C883-C89F	ESEGUE GOSUB
C8A0-C8D1	ESEGUE GOTO
C8D2-C8EA	ESEGUE RETURN
C8EB-C905	ESEGUE DATA
C906-C908	RICERCA DEL PROSSIMO COMANDO BASIC
C909-C927	RICERCA LA PROSSIMA LINEA BASIC
C928-C93A	ESEGUE IF
C93B-C94A	ESEGUE REM
C94B-C96A	ESEGUE ON
C96B-C9A4	RIPORTA UN NUMERO FIXED POINT DAL BASIC
C9A5-CA1C	ESEGUE LET (PARTE PRIMA DELLA ROUTINE)

CA1D-CA2B	AGGIUNGE UN DIGIT ASCII ALL' ACCUMULATORE N 1
CA2C-CA7F	ESEGUE LET (PARTE SECONDA)
CA80-CA85	ESEGUE PRINT£
CA86-CA99	ESEGUE CMD
CA9A-CB1D	ESEGUE PRINT
CB1E-CB3A	STAMPA UNA STRINGA DALLA MEMORIA
CB3B-CB4C	STAMPA UN CARATTERE IN SINGOLO FORMATO (spazio,?)
CB4D-CB7A	CONTROLLA UN INPUT ERRATO
CB7B-CBA4	ESEGUE GET
CBA5-CBBE	ESEGUE INPUT No
CBBF-CBF8	ESEGUE INPUT
CBF9-CC05	PREPARA PER RICEVERE INPUT
CC06-CCFB	ESEGUE READ
CCFC-CD1D	EXTRA IGNORED E REDO FROM START
CD1E-CD77	ESEGUE NEXT
CD78-CD9D	CONTROLLA I DATA,STAMPA TYPE MISMATCH
CD9E-CEF0	INPUT E VALUTAZIONE DI ESPRESS. NUMERICHE O NON
CEF1-CEF6	VALUTA LE ESPRESSIONI DENTRO LE ()
CEF7-CEF9	CONTROLLO PARENTESI DESTRE
CEFA-CEFC	CONTROLLA PARENTESI SINISTRE
CEFD-CF07	CONTROLLO DELLA VIRGOLA
CF08-CF0C	STAMPA SYNTAX ERROR E ESCE
CF0D-CF13	LIBERO PER FUTURE VALUTA DI FUNZIONI
CF14-CFA6	CERCA IL NOME DELLA VARIABILE
CFA7-CFE5	IDENTIFICA I RIFERIMENTI DELLE FUNZIONI
CFE6-CFE8	ESEGUE OR
CFE9-D015	ESEGUE AND
D016-D07D	ESEGUE CONFRONTI, NUMERICI O STRINGA
D07E-D08A	ESEGUE DIM
D08B-D112	RICERCA LA LOCAZIONE DI UNA VARIABILE IN MEMORIA
D113-D11C	CONTROLLA SE UN CARATTERE ASCII E' ALFABETICO
D11D-D193	CREA UNA NUOVA VARIABILE BASIC
D194-D1A4	SUBROT. PER PUNTATORI DI MATRICI
D1A5-D1A9	32768 IN BINARIO
D1AA-D1D0	VALUATZIONE DI ESPRESSIONE PER INTERO POSITIVO
D1D1-D34B	TROVA O CREA UNA MATRICE
D34C-D37C	CALCOLO DELLA MATRICE
D37D-D390	ESEGUE FRE
D391-D39D	CONVERTE FIXED POINT IN FLOATING POINT
D39E-D3A5	ESEGUE POS
D3A6-D3B2	CONTROLLA SE E' UN COMANDO DIRETTO POSSIBILE

D3B3-D3E0	ESEGUE DEF
D3E1-D3F3	CONTROLLA LA SINTASSI DI FN
D3F4-D464	VALUATA FN
D465-D474	ESEGUE STRö
D475-D486	CALCOLA IL VETTORE STRINGA
D487-D4F3	CONTROLLA LA STRINGA
D4F4-D525	SUB. DI COSTRUZIONE VETTORE STRINGA
D526-D5BC	ROUTINE DELLA GARBAGE COLLECTION
D5BD-D605	CONTROLLO DELLA STRING COLLETION
D606-D63C	COLLECT STRING
D63D-D679	ESEGUE LA CONCATENAZIONE DI STRINGHE
D67A-D6A2	METTE UNA STRINGA IN MEMORIA
D6A3-D6DA	SCARTA UNA STRINGA NON CERCATA
D6DB-D6EB	PULOSCE LO STACK DESCRIPTOR
D6EC-D6FF	ESEGUE CHRö
D700-D72B	ESEGUE LEFTö
D72C-D736	ESEGUE MIDö
D761-D77B	CARICA I PARAMETRI DI UNA STRINGA
D77C-D781	ESEGUE LEN
D782-D78A	PASSA DA MODO STRINGA A MODO NUMERICO
D78B-D79A	ESEGUE ASG
D79B-D7AC	INPUT PER I PARAMETRI DEL BYTE
D7AD-D7EA	ESEGUE VAL
D7EB-D7F6	DA DUE PARAMETRI PER POKE O WAIT
D7F7-D80C	CONVERTE IL FLOATING POINT IN FIXED POINT
D80D-D823	ESEGUE PEEK

D824-D82C	ESEGUE POKE
D82D-D848	ESEGUE WAIT
D849-D84F	AGGIUNGE .5 ALL' ACCUMULATORE N 1
D850-D861	ESEGUE LA SOTTRAZIONE
D862-D946	ESEGUE ADDIZIONE
D947-D97D	COMPLEMENTO ALL' ACCUMULATORE N 1
D97E-DP82	STAMPA OVERFLOW E ESCE
D983-D9BB	SUBROU DI MULTIPLY A BYTE
D9BC-D9E9	COSTANTI NELLE FUNZIONI
D9EA-DA2F	ESEGUE LOG
DA30-DA58	ESEGUE LA MOLTIPLICAZIONE
DA59-DA8B	SUBROUTINE DI MOLTIPLICAZIONE BIT
DA8C-DAB6	CARICA L' ACCUMULATORE N 2
DAB7-DAD3	TEST E AGGIUSTAGGIO DEGLI ACCUMULATORI 1 E 2
DAD4-DAE1	CONTROLLO DI OVERFLOW E UNDERFLOW
DAE2-DAF8	MOLTIPLICA PER 1
DAF9-DAFD	10 IN FLOATYNG BINARIO
DAFE-DB06	DIVIDE PER 10
DB07-DB11	ESEGUE DIVIDE ENTRO
DB12-DBA1	ESEGUE DIVIDE PER
DBA2-DBC6	CARICA L' ACC. N 1 DALLA MEMORIA
DBC7-DBFB	IMMAGAZZINA IN MEMORIA L'ACC. N 1
DBFC-DC0B	COPIA L'ACCUMULATORE N 2 NELL 'ACC. 1
DC0C-DC1A	COPIA L'ACCUMULATORE N 1 NELL' ACC. 2
DC1B-DC2A	ARROTONDA L'ACCUMULATORE N 1
DC2B-DC38	CALCOLA IL VALORE SGN DELL' ACC. N 1
DC39-DC57	ESEGUE SGN
DC58-DC5A	ESEGUE ABS
DC5B-DC9A	CONFRONTA L' ACCUMULATORE 1 CON LA MEMORIA
DC9B-DCCB	CONVERTE UN NUMERO IN FLOATING IN FIXED POINT
DCCC-DCF2	ESEGUE INT
DCF3-DD7D	CONVERTE UNA STRINGA IN FLOATING POINT
DD7E-DDB2	DA UN NUOVO DIGIT ASCII
DDB3-DDC1	COSTANTI DI CONVERSIONE STRINGA
DDC2	STAMPA IN
DDCD-DDDC	STAMPA IL NUMERO DI LINEA BASIC
DDDD-DF10	CONVERTE UN NUMERO O TIö IN ASCII
DF11-DF70	COSTANTI PER CONVERSIONE NUMERICA
DF71-DF77	ESEGUE SQR
DF78-DFB3	ESEGUE UNA FUNZIONE POTENZA
DFB4-DFBE	ESEGUE NEGAZIONE BOLEANA

DFBF-DFEC	COSTANTI PER VALUTAZIONE STRINGA
DFED-E03F	ESEGUE EXP
E040-E089	SUBROT.NES DI VALUATZIONE DI FUNZIONI
E08A-E093	MANIPOLAZIONI DI COSTANTI PER RND
E094-E0F5	ESEGUE RND
E0F6-E260	ROUTINES DI KERNAL
E261-E267	ESEGUE COS
E268-E2B0	ESEGUE SENO
E2B1-E2DC	ESEGUE TANGENTE
E2DD-E30A	COSTANTI PET VALUTAZIONE TRIG
E30B-E33A	ESEGUE ATN
E33B-E377	COSTANTI PER VALUTAZIONI SERIE DI ATN
E378-E386	INIZIALIZZA I VETTORI RAM
E387-E3A3	SUBROUTINE PER RINVIO IN PAGINA ZERO
E3A4-E428	INIZIALIZZAZIONE DEL SISTEMA
E429-E44E	BYTES FREE ßßßß CBM BASIC V2ßßßß
E44F-E47B	VETTORI DI INIZIALIZZAZIONE
E47C-E4FF	SPAZIO NON USATO

VALORI DEI CODICI BASIC

cod	caratt.	cod	caratt.	cod	caratt.	cod	caratt.
0	fine lin.	66	B	133	input	169	step
1-31	non usati	67	C	134	dim	170	+
32	spazio	68	D	135	read	171	-
33	!	69	E	136	let	172	*
34	"	70	F	137	goto	173	/
35	£	71	G	138	run	174	
36	$	72	H	139	if	175	and
37	%	73	I	140	restore	176	or
38	&	74	J	141	gosub	177	
39	'	75	K	142	return	178	=
40	(	76	L	143	rem	179	
41	)	77	M	144	stop	180	sgn
42	*	78	N	145	on	181	int
43	+	79	O	146	wait	182	abs
44	,	80	P	147	load	183	usr
45	-	81	Q	148	save	184	fre
46	.	82	R	149	verify	185	pos
47	/	83	S	150	def	186	sqr
48	0	84	T	151	poke	187	rnd
49	1	85	U	152	print£	188	log
50	2	86	V	153	print	189	exp
51	3	87	W	154	cont	190	cos
52	4	88	X	155	list	191	sin
53	5	89	Y	156	clr	192	tan
54	6	90	Z	157	cmd	193	atn
55	7	91	°	158	sys	194	peek
56	8	92		159	open	195	len
57	9	93		160	close	196	str$
58	:	94		161	get	197	val
59	;	95		162	new	198	asc
60		96		163	tab(	199	chr$
61	=	128	end	164	to	200	left$
62		129	for	165	fn	201	right$
63	?	130	next	166	spc(	202	mid$
64		131	data	167	then	203-254	non u.
65	A	132	input	168	not	255	

$C43A
Questa routine fornisce i messaggi di errore ricavandoli dalla relativa TAVOLA che ha come indirizzo $c193.
Il numero del messaggio e' contenuto nel registro indice X.
Il messaggio e' evidenziato sul device aperto in quel momento per l'Output, di solito il video.
Puo' essere usata per generare in forma rapida dei messaggi di errore durante l'esecuzione di programmi per quanto sia limitata dalla Tavola Standard.

$c483
Questa routine accetta una nuova linea Basic da tastiera e ne esegue il contenuto in modo diretto oppura la immagazzina in modo indiretto.
Puo' essere utilizzata per aggiungere comandi extra al Basic.

$c560
Le stringhe di datifino a 88 caratteri di lunghezza sono ricevute da questa routine ed immaggazzinati in una locazione di memoria RAM chiamata BASIC INPUT BUFFER.
Questa locazione di memoria va da $0200 a $0258. Il termine della stringa e' segnalato da un BYTE a ZERO

$c57c
Attraverso questa routine i comandi Basic composti da 3 a 7 lettere, sono convertiti in una lunghezza di 1 solo byte ottimizzandone quindi l' immagazzinamento in memoria.
Anche questa routine puo' essere utile per aggiungere comandi al Basic.

$cble
Si utilizza per stampare una stringa sulla periferica aperta in output (di solito lo schermo).
L' indirizzo di memoria corrispondente all' inizio della stringa e' definito dal contenuto del Registro indice Y (LSB dell' indirizzo) e dall' Accumulatore (MSB dell' indirizzo).
La fine della stringa da stampare e' dato dal primo Byte contenente zero , in binario, che si incontra.

$ce86
Questa routine valuta un' espressione Bsic che inizia dall' indirizzo definito dal contenuto delle locazioni $7a e $7b.
Il risultato dell' espressione e' immagazzinato nell' accumulatore n 1.

$cfe6
Viene effettuato un OR logico fra i valori contenuti nell' Accumulatore n 1 e n 2 ed il risultato viene immagazzinato nell' Accumulatore n 1

$cfeb
Funzionamento come per la routine anzidetta solo che esegue un AND logico.

$d1aa
Tramite questa routine un numero immagazzinato nell' Accumulatore n 1 sotto forma di Floating Point e' convertito in Integer ed immesso nei due Bytes $64 e $65 sempre dell' Accumulatore n 1.
Il formato dell' Integer e' 100x$64+$65.

$d37d
Questa funzione determina il numero di Bytes liberi della memoria utilizzabili per dati o programmi.
L' argomento della funzione viene immagazzinato come numero in Floating Point nell' Accumulatore n 1.

$d391
Il valore di un numero Integer contenuto nel Registro indice Y e nell' Accumulatore e' convertito tramite questa rotine in Floating Point e immagazzinato nell' Accumulatore n 1.

$d77c

Questa routine serve per calcolare la lunghezza di una stringa, cioe' il numero di caratteri che la compongono.
L' argomento della funzione, ad esempio il nome della stringa e' immagazzinato nei Bytes $64 e $65 dell' Accumulatore n 1.
La lunghezza della stringa e' riportata nel Registro indice X.

$d850

Tramite questa routine i contenuti dell' Accumulatore n 2 sono sottratti dai contenuti dell' Accumulatore n 1 ed il risultato inserito nell' Accumulatore n 1.
Prima che sia chiamata questa Routine il segno di confronto del Byte $6f deve essere settato. Questa operazione viene esegutia tramite un OR esclusivo fra i contenuti delle locazioni di memoria $66 e $ 6e ed il risultato immagazzinato in $6f.
L' Accumulatore di processo dovrebbe contenere il valore immagazzinato in $61 (MSB dell' Accumulatore n 1).

$d9ea

Questa routine definisce la funzione LOG.
Il valore usato nell'argomento di questa funzione e' contenuto nell' Accumulatore n1 ed il risultato e' posto nello stesso Accumulatore.

$da30
Questa routine per prima cosa ricerca il contenuto dell' Accumulatore n2 dalla memoria.
La locazione di memoria e' identificata tramite un indirizzo di due Bytes contenuti nell' Accumulatore di processo e nel registro indice Y.
Il formato usato e' 100xindice Y+Accumulatore di processo.
Il valore immagazzinato nell' Accumulatore n 2 e' allora moltiplicato per il contenuto dell' Accumulatore n 1 ed il risultato sempre immagazzinato nell'Accumulatore n1.

$da33
Il contenuto dell' Accumulatore n 2 e' moltiplicato per il contenuto dell' Accumulatore n 1 ed il risultato immesso nell' Accumulatore n 1.
Prima di usare questa routine il segno di confronto contenuto nel Byte $6f deve essere settato il che si ottiene eseguendo un OR esclusivo fra i contenuti dei Bytes $66 e $6e ed immagazzinando appunto il risultato in $6f.
L' esponente del valore contenuto nell' Accumulatore n 1($61) dve essere caricato nell' accumulatore di processo prima di far girare questa routine.

$da8c
Questa routine prende un dato valore immagazzinato in memoria e lo carica nell' Accumulatore n 2.
I due Bytes di memoria che danno l'indirzzo del valore detto sono inseriti nell'Accumulatore di processo e nel Registro indice Y, mentre il formato usato e':
Accumulatore +100*registro indice Y.

$dae2

Il contenuto dell' Accumulatore n 1 viene moltiplicato per 10 ed il risultato immesso nell' Accumulatore n 2.

$dafe
Il contenuto dell' Accumulatore n 1 e' diviso per 10 ed il risultato immagazzinato nell' Accumulatore n 2.

$db0f
Questa routine divide il contenuto dell' Accumulatore n2 per il contenuto dell' Accumulatore n1 ed immette il risultato nell' Accumulatore n1.
Prima di far girare questa routine deve essere settato il segno del Byte di $6f, eseguendo un OR esclusivo fra i segni contenuti nelle due locazioni di memoria $66 e $6e ed immettendo il risultato in $6f.
L' esponente del valore dell' Accumulatore n 1, contenuto in $ 61, dovrebbe essere caricato nell' Accumulatore di processo prima di far girare questa routine.

$dba2
Questa routine prende un dato valore dalla memoria e lo carica nell' Accumulatore n 1.
Per l' esecuzione ed i modi vedi quanto detto nella routine $da8c.

$dbc7
Tramite questa routine il valore dell' Accumulatore n 1 e' immagazzinato in una specificata locazione di memoria.
L' indirizzo di memoria specificato di due Bytes e' nei registri indice di processo X e Y, mentre il formato usato e:
Indice X+100*registro indice Y.

$dbfc
Il valore attuale dell' Accumulatore n 2 e' copiato integralmente nell' Accumulatore n 1.
Il valore dell' Accumulatore n 2 resta inalterato.

$dc0c
Il contenuto dell' Accumulatore n 1 e' copiato nell' Accumulatore n 2.
Il contenuto dell' Accumulatore n 1 e' fatto girare e se necessario il suo esponente aggiustato.

$dc0f
Come per la routine precedente solo che si riferisce all' altro accumulatore.

$dc58
Il valore assoluto contenuto nell' Accumulatore n 1 e' riportato appunto in assoluto nello stesso accumulatore.

$dc39
Questa routine riporta il segno del valore immagazzinato nell' Accumulatore n 1.
Se il valore dell' Accumulatore n 1 e' maggiore di 0 allora nello stesso accumulatore viene immagazzinato il valore 1, se e' uguale a 0 allora viene messo a 0, se e' minore di 0 viene segnato come -1.

$dc5b
Il valore corrente dell' Accumulatore n 1 e' confrontato con il valore immagzzinato in memoria e l' Accumulatore di processo si setta a questo valore in di pendenza se queste due variabili sono

eguali o diverse.
L' indirizzo della memoria composto da due Bytes e' immagazzinato nell' Accumulatore di processo e nel Registro indice Y.
Il formato usato e' Accumulatore di processo +100*Registro indice Y.
Se le due variabili sono eguali allora l' Accumulatore di processo e' settato a $00 in caso contrario a $ff.

$dc9b
Una variabile in Floating Point immagazzinata nell' Accumulatore n 1 viene convertita in formato Fixed point tramite questa routine.
Il valore trovato viene poi immesso nell' Accumulatore n 1.

$dccc
Questa routine converte una variabile in Floating point immagazzinata nell' Accumulatore n 1 in un valre intero.
Il risultato viene inserito nell' Accumulatore n 1.

$df71
Questa routine serve per calcolare la radice quadrata di una variabile in Floating Point inserita nell' Accumulatore n1 e reinserire il risultato nello stesso accumulatore.

$e094
Questa routine serve per creare un numero casuale che viene immesso nell' Accumulatore n 1.
E' da sottolineare che le locazioni di memoria da $8b a $90 contengono l' ultimo numero casuale generato.

$e261
Questa routine serve per calcolare il COSENO di un valore espresso in radianti e caricato nell' Accumulatore n 1.
Il risultato del calcolo e' immesso nell' Accumulatore n 1.

$e268
Questa routine serve per calcolare il SENO di un valore espresso in radianti e caricato nell' Accumulatore n 1.
Il risultato di calcolo e immesso nello stesso Accumulatore.

$e2b1
Stesso funzionamento che per le precedenti solo che il calcolo avviene per la TANGENTE.

$e30b
Stesso funzionamento ma per la funzione ARCOTANGENTE.

$e378
Questa routtine serve per inizzializzare tutto il sistema di vettoroi e di variabili.
Puo' essere usata quando si intenda reinizzializzare il sistema senza spegnere la macchina.
In particolare si usa quando si desideri ritornare al Basic provenienti da un Programma in Linguaggio macchina che ha alterato alcuni valori.

LE ROUTINES KERNAL DEL VIC

Il sistema operativo del VIC e' stato disegnato in maniera particolare per consentire un facile accesso a queste Routines.
Queste subroutines possono esser usate tramite programmi in Linguaggio macchina che le richiamano attraverso degli indirizzi RAM.
Il gruppo piu' importante e' collocato nella parte piu' alta della memoria ROM, mentre gli indirizzi delle stesse sono individuabili nella terza pagina di memoria RAM.

La ragione per la quale questi indirizzi di vettori di salto sono collocati in una parte RAM della memoria dipende dal fatto che questi possono cosi' essere facilmente cambiati e manipolati.

Diamo qui di seguito l'elenco di queste subroutines con il relativo modo d' uso e funzionamento.

TAVOLA DELLE ROUTINE KERNAL

E500-E504	RIPORTO INDIRIZZO DEL 6522
E505-E509	RIPORTA IL MASSIMO DELLE RIGHE E COLONNE
E50A-E517	LEGGE/STAMPA LA POSIZIONE DEL CURSORE
E518-E580	INIZIALIZZA I/O
E581-E586	HOME FUNCTION
E587-E5B4	MUOVE IL CURSORE SECONDO IL PUNTATORE
E5B5-E5C2	ENTRATA DEL NMI (TASTO RESTORE)
E5C3-E5CE	INIZIALIZZA IL 6561
E5CF-E64E	TOGLIE IL CARATTERE DALLA CODA
E64F-E741	INPUT DI LINEA DOPO RETURN
E742-E8E7	STAMPA DI ROUTINE
E8E8-E8F9	CONTROLLO DECREMENTO NEL PUNTATORE DI LINEA
E8FA-E911	CONTROLLO INCREMENTO NEL PUNTATORE DI LINEA
E912-E928	CONTROLLO COLORE
E929-E974	TAVOLA DI CONVERSIONE DA CODICE SCHERMO A ASCII
E975-EAA0	ROUTINES DI SCROLL DELLO SCHERMO
EAA1-EB1D	ROUTINE DI IRQ
EB1E-EC45	ANALISI GENERALE DI TASTIERA
EC46-EE13	TAVOLE DELLE MATRICI PER LA TASTIERA
EE14-EEBF	COMANDO AL BUS SERIALE DELLA PERIFERICA IN ATTESA
EEC0-EEC4	INVIA L'INDIRIZZO SECONDARIO DOPO L'ASCOLTO
EEC5-EECD	RILASCIA LINEA ATTENTION DOPO L'ASCOLTO
EECE-EEE3	PARLA L' INDIRIZZO SECONDARIO
EEE4-EEF5	USCITA POTENZIATA AL BUS SERIALE
EEF6-EF03	MANDA UN COMANDO UNTALK AL BUS SERIALE
EF04-EF18	MANDA UN OMANDO UNLISTEN AL BUS SERIALE
EF19-EFA2	INPUT DI UN BYTE DAL BUS SERIALE
EFA3-EFED	ROUTINE CONTINUA DI NMI
EFEE-F035	TRASMETTE UN BYTE
F036-F173	ROUTINE NMI PER RAGGRUPPARE I DATA NEI BYTES
F174-F1E1	MESSAGGI KERNAL
F1E2-F1F4	SCRIVE UN MESSAGGIO SULLO SCHERMO
F1F5-F20D	PRENDE UN CARATTERE DA UN CANALE
F20E-F279	INPUT DI UN CARATTERE DAL CANALE
F27A-F2C6	USCITA DI UN CARATTERE A UN CANALE
F2C7-F308	APRE UN CANALE PER INPUT
F309-F349	APRE UN CANALE PER OUTPUT

F34A-F3EE	CHIUDE UN FILE LOGICO
F3EF-F3F2	CHIUDE TUTTI FILES LOGICI
F3F3-F409	PULISCE I CANALI (CLEAR)
F40A-F541	FUNZIONE DI APERTURA
F542-F674	CARICA FUNZIONI RAM
F675-F733	SAVE FUNZIONI RAM
F734-F76F	FUNZIONE TIME
F770-F77D	CONTROLLA IL TASTO DI STOP
F77E-F7AE	MANOVRA DI ERRORE
F7AF-F889	CERCA E LEGGE LA TESTATA DEL NASTRO
F88A-F98D	ROUTINE DI CONTROLLO CASSETTA
F98E-FABC	ROUTINE DI LETTURA NASTRO
FABD-FBE9	OPERAZIONE SU UN BYTE PER LETTURA NASTRO
FBEA-FD21	ROUTINE DI SCRITTURA NASTRO
FD22-FE90	INIZIALIZZA IL SISTEMA ALL'ACCENSIONE
FE91-FEA8	ROUTINE DI CONTROLLO MEMORIA
FEA9-FF5B	MANOVRA L' NMI
FF5C-FF71	TAVOLE DI BAUD RATE
FF72-FF85	MANOVRA L'IRQ
FF86-FFFF	INDIRIZZI DEI VETTORI DI SALTO KERNAL

V I C   K E R N A L   R O U T I N E S

$ff8a
Ristabilisce i valori saltati dei vettori per il sistema di subroutines e gli interrupts.

$ff8d
Se questa routine e' chiamata quando il bit del Carry e' settato, verra' allora letto l' attuale contenuto dei vettori RAM e immagazzinato in una lista che inizia ad una locazione di memoria il cui indirizzo e' contenuto in X e Y.
Quando questa routine e' chiamata con il bit di carry in CLEAR, la lista utente il cui indirizzo e' determinato dai puntatori in X e Y, allora la lista e' trasferita nei vettori RAM.
Quando si usa questa routine il miglior modo e' di leggere prima l' insieme dei vettori, di trasferirli in un' area di memoria libera, modificarli e poi rimettrli nel sistema.

$ff90
Questa routine controlla la stampa degli errori ed i messagi diagnostici.
E' chiamata immettendo un valore nell' accumulatore di processo

$ff93
Manda un indirizzo secondario dopo un LISTEN della routine $ffb1.
Questa Routine non puo' pero' essere usata dopo un comando di TALK proveniente dall' uso della routine $ffb4

$ff96
Manda un indirizzo secondario per un comando TALK.
Caricando l' Accumulatore di processo con un numero tra 0 e 31 con questa routine si invia un comando di indirizzo secondario sulla IEEE.
Questa routine puo' essere usata solo dopo la COMMAND IEEE DEVICE TO TALK contenuta in $ffb4. Non puo' essere usata dopo la routine di posizione $ffb1 COMMAND IEEE DEVICE TO LISTEN.

$ff99
Quando si richiama questa routine ed il bit di carry e' settato, allora il puntatore al punto piu' alto della memoria RAM e' letto nei registri indice Y eX.
Quando invece si richiama ed il bit di carry e' in CLEAR allora il contenuto dei registri indice Y e X sara' copiato nel puntatore detto.

$ff9c
Viene eseguita la funzione descritta nella precedente routine pero' sul puntatore della parte piu' bassa della memoria.
Ricordare che il valore in questo caso e nell' indirizzo $0400

$ff9f
Questa routine esegue lo scanning , cioe' la lettura, della tastiera.
Se un tasto e'premuto allora il suo valore (o meglio il valore corrispondente al codice ASCII relativo) e' immesso in quella particolare posizione di memoria chiamata KEYBOARD QUEUE (coda di tastiera) fra $0277 e $0280.
Questa e' la stessa routine chiamata dall' INTERRUPT HANDLING ROUTINES una volta ogni sessantesimo di secondo.

$ffa2
Quando l' accumulatore del processore contiene uno 0 nel bit 7 questa routine abilita la temporizzazione esterna, mentre con il bit 7 a 1 la temporizzazione viene disabilitata.
La temporizzazione e' il metodo mediante il quale il VIC puo' colloquiare con una periferica senza andare in una sequenza di TIMESHAKE. La periferica deve rispondere al DAV entro 64 ms.
I dischi del Vic e del Cbm usano la temporizzazione per comunicare uno stato di 'file not found' dopo un comando open.

$ffa5
Questa routine attende un byte di risposta dal bus IEEE. Il dato e' caricato nell' accumulatore.
Si conviene che la periferica sia in 'talk' per mezzo della routine $ffb4 con l' esistenza di un eventuale indirizzo secondario chiamato dalla routine $ff96.

IL VIC ED IL PET

Il motivo per cui e' stato scritto questo capitolo e' che esistono numerosi programmi e procedure sul PET o comunque sulle linee CBM, mentre ancora relativamente pochi per il VIC 20, ragion per cui crediamo opportuno di dare alcune nozioni circa la possibilita' esistente di trasferire programmi dal PET al VIC o viceversa.
Ci preme comunque di sottolineare fin d' ora che questo lavoro va eseguito con le opportune cautele.
Il problema di trasferire un programma PET o CBM e di farlo girare su un VIC puo' essere diviso in due parti.
Dovremo fisicamente muovere un programma da una macchina ad un altra, secondariamente dovremo modificare il programma trasferito affinche' possa funzionare.
Se il programma e' in BASIC allora il compito e' relativamente facile, se invece e' stato scritto in Assembler o in altro linguaggio allora il lavoro sara' relativamente piu' complesso.
Ci sono tre possibili strade per trasferire i programmi dal PET al VIC o (viceversa naturalmente):

CASSETTA

DISCO

COMUNICAZIONE DIRETTA

Vediamoli uno per uno.

USO DELLA CASSETTA -Il formato dell' unita' a cassetta per il PET e per il VIC e' esattamente lo stesso per cui un VIC puo' leggere un programma scritto su un nastro PET o viceversa senza alcun problema.

USO DEI DISCHI - Il di sco del VIC, cioe' il 1540 CBM, ha un suo particolare formato di memorizzazione ed e' un formato differente da quello usato nei drive del PET.
Il formato del 1540 e' compatibile in lettura con il 2040/3040/4040. Questo consente di poter leggere i dischi 2040/3040/4040 in un drive del CBM 1540 e si possa leggere i dischi 1540 su i drive 2040/3040/4040.
Non e' invece possibile l' operazione di scrittura. In pratica si verifica questo fenomeno si puo' caricare un programma da un disco PET ma successivamente questo andra' riscritto su di una unita' 1540.
Sfortunatamente questo discorso non e' valido per le unita' a Floppy 8050 e 8250 per cui se si possiede dei programmi scritti su questi tipi di unita' dovremo cercare un' altra strada di comunicazione.

USO DI UN COMMUNICATION LINK - Ci sono numerosi costruttori, oltre alla COMMODORE, che offrono delle interfacce RS232 per il VIC, come ci sono numerose interfacce per il PET.
Usando pero' questo sistema di collegamento si richiede anche una piccola serie di sottoprogrammi o SUBROUTINES la cui codifica dipende soprattutto dal tipo di interfaccia che si usa.
Nel caso pertanto che si disponga di una interfaccia del tipo RS232 sara' necessario guardarla dalla parte del VIC per vedere se e' un' interfaccia piena o un' interfaccia seriale VIC RS232 (per questo vedere il capitolo dedicato all' interfaccia RS 232 del VIC)inoltre sara' necessario osservarla dalla parte del PET ed esaminare accuratamente i tipi di istruzioni che le case produttrici di queste o di altri tipi di interfacce forniscono.

La seconda fase e' quella di far girare, una volta

caricati i programmi dal PET sul VIC, questi programmi.
La piu' grossa differenza nell' uso di queste due macchine e' nell' uso della memoria per i programmi, del BASIC e della memoria e della gestione di schermo.
Il formato attuale per i programmi e' lo stesso per entrambi, ma la Pagina Zero e' diversa come pure diverse sono le ROM e di conseguenza alcune cose del Sistema Operativo.
Questo ci fa capire subito che qualsiasi programma che usi un gruppo di istruzioni PEEK e POKE avra' delle difficolta' ad essere convertito. Riportiamo in fondo a questo articolo la mappa di memoria del PET per una maggiore comprensione.
Una caratteristica interessante del VIC e' il fatto che la mappa di memoria cambia qualora si aggiungano moduli di espansione e quindi si passi ad un diverso numero di Bytes disponibili.
Cio' non ha nessun effetto sui programmi Basic a parte i comandi di PEEK e POKE ma i programmi scritti in Assembler, cambieranno in maniera sostanziale.
Riportiamo qui sotto una tavola degli indirizziBasic e dell' area di memoria schermo rispetto alla memoria presente nel VIC:

AREA DI MEMORIA	5K	5k+3k	5k+8k
Inizio progr. basic	4096	1024	4608
RAM schermo	7608-8186	7680-8186	4096--4602
RAM colore	38400-38906	38400-38906	37888-38394

C'e una piccola differenza nel sistema in cui un programma e' immagazinato su disco o su nastro tra il PET e il VIC.
Sul PET il File contiene un indirizzo che determina il punto di memoria in cui verra' caricato il programma.
Per il Basic questo indirizzo il decimale 1025, ma un programma in linguaggio macchina potrebbe anche essere caricato in altro punto.
Per il VIC questi indirizzi non sono incompatibili, ma semplicemente non sono usati.
Nel VIC infatti un programma viene caricato sempre all' indirizzo dato nella tavola, e questo significa che se si deve caricare un programma in linguaggio macchina a partire dal punto massimo della memoria o del buffer del nastro, dovremo scrivere una routine per muoverlo dal punto massimo della memoria al punto desiderato.
Si intende cioe' dire che questo non viene caricato automaticamente come sul PET.
Ci sono anche delle differenze di schermo fra le due macchine, cosi' che quasi tutti i comandi di PRINT devono essere cambiati.
Ricordiamo inoltre che i comandi di POKE sullo schermo dovranno essere cambiati uno per uno.

Area	Indirizzo
AREA LAVORO S.O E BASIC	0 1024
AREA RAM UTENTE PER SISTEMI DA 8 K A 32 K	32768
AREA RAM PER MEMORIA DI SCHERMO E VIDEO	36864
AREA DI ESPANSIONE ROM 12 K	49152
AREA ROM PER BASIC	59392
I/O	61440
AREA ROM PER S.O	65535

MAPPA DI MEMORIA DEL PET

CONVERSIONE DA PET AL VIC

Questa e' probabilmente la piu' facile delle due opzioni poiche' un programma PET non avra' ne' comandi relativi ai suoni ne' comandi relativi ai colori per cui passandolo sul VIC dovremo aggiungerli comunque.
Il cambiamento maggiore sara' quello relativo allo schermo.
Il VIC ha uno schermo di 23 linee ciascuna delle quali di 22 caratteri, mentre il PET ha 25 linee di 40 o di 80 caratteri.
Il fatto che un programma BASIC occupi attualmente una differente posizione di memoria non deve preoccupare in quanto il VIC carichera' SEMPRE questo programma nella corretta posizione di memoria.
Una cosa che bisogna controllare accuratamente e' che sul VIC ci sia sufficiente memoria per caricare e per far girare il programma proveniente dal PET.

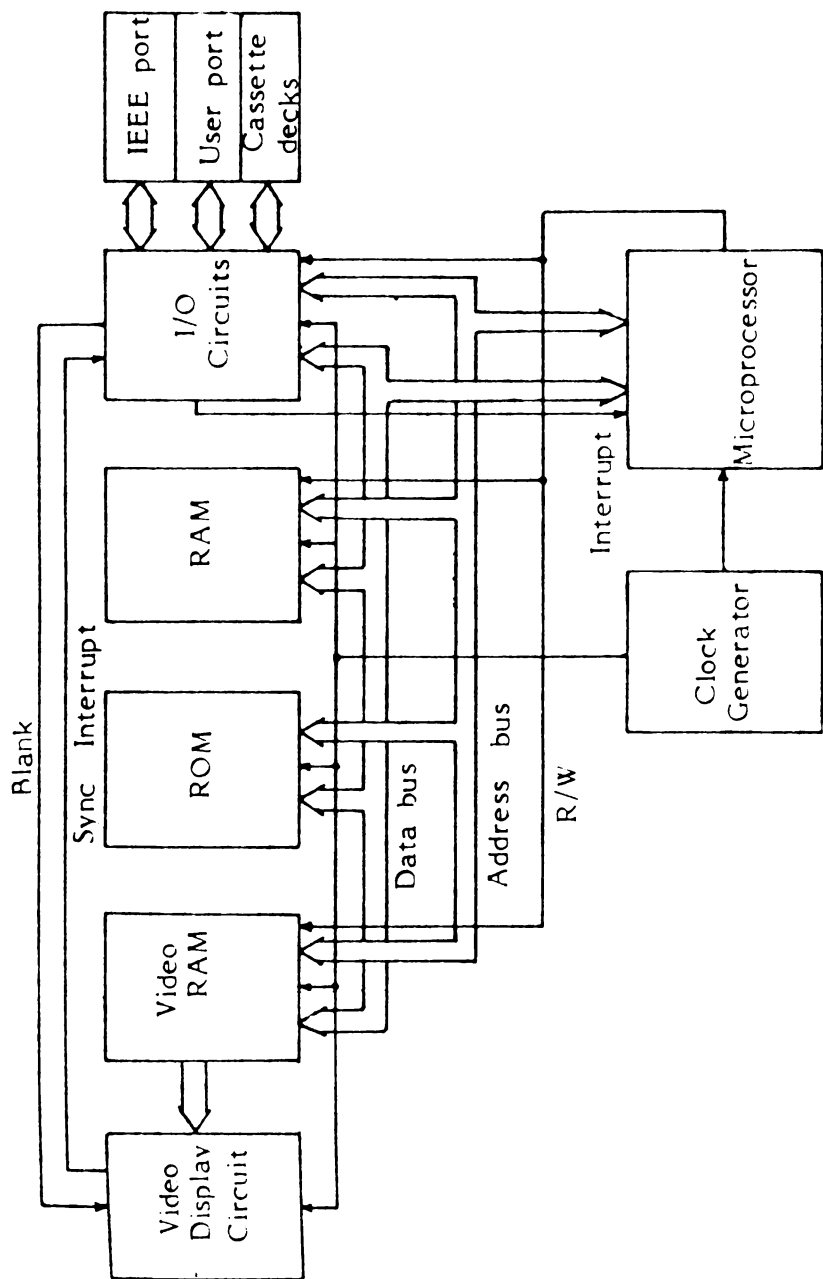
NB. A rischio di ripeterci vogliamo sottolineare che i programmi in Linguaggio Macchina o i programmi in BASIC che contengano numerosi comandi di PEEK o POKE avranno una notevole difficolta' ad essere convertiti.

CONVERSIONE DAL VIC AL PET

Come abbiamo gia' detto prima ogni programma relativo allo schermo e' stato scritto per essere differente fra il VIC ed il PET. Usando solo comandi PRINT ma lo schermo o le indicazioni che appariranno sullo schermo saranno spostate.
Se invece si sta usando applicazioni particolari di colore o di suono sul VIC queste dovrebbero essere tolte prima di provare a far girare il programma sul PET.Il PET quando carica dei programmi che sono stati scritti per il VIC " RITIENE" che i suoi programmi Basic abbiano inizio alla locazione 1025, ma questo avviene solo per i VIC che siano espansi ad 8 K-Bytes, mentre nel caso che il VIC sia in configurazione base o sia oltre gli 8 K-Bytes sara' necessario eseguire le modifiche riportate nella sottostante tabella:

PER VIC CON MEMORIA DA

5 K	OLTRE 8 K
1 Caricare il programma	
2 Inserire una nuova linea: 10 rem	
3 Inserire i comandi:	
POKE 1025.7 POKE 1026,16	POKE 1025.7 POKE 1026,18
4 Cancellare linea 10	
5 Eseguire:	
POKE 43,(PEEK(43)-12) CLR	POKE 43,(PEEK(43)-14):CLR

E' necessario avere molta cura nell' eseguire la procedura riportata nella tabella.
Bisogna ricordare infatti che la linea n. 10 di questa procedura deve contenere solo il numero di linea ed il comando REM. E' chiaro che se il programma sul VIC dovesse iniziare con una linea n. 10 o comunque essere presente una linea con questo numero, si potra' dare un qualsiasi altro numero di linea purche' minore di 10.
Da evitare ASSOLUTAMENTE di listare il programma prima di aver eseguito la procedura descritta.

STRUTTURA DI UN MICROELABORATORE

STRUTTURA DI UN MICROELABORATORE

Premessa

La parte relativa al linguaggio macchina, alla gestione quindi del microprocessore ed all' uso specializzato delle relative Routines, non puo' prescindere da una conoscenza approfondita dell' integrato 6502, delle tecniche assembler, ecc.
Poiche' questo avrebbe richiesto di scrivere un volume separato, come infatti stiamo facendo, abbiamo in questo manuale dovuto per causa di spazio ad una enunciazione a grandi linee rimandando per una trattazione piu' approfondita al manuale:

IL 6502 ED IL SUO USO
che stiamo appunto preparando.

Un microelaboratore e' un sistema elettronico in grado di ricevere dall' esterno segnali elettrici, immagazzinarli, elaborarli secondo un certo programma, prendere delle decisioni e di conseguenza emettere dei segnali elettrici utilizzabili all' esterno.

Essenzialmente un microelaboratore e' composto da cinque grandi blocchi funzionali:

CPU (Central Processing Unit)
Unita' centrale di elaborazioe

RAM (Random Access Memory)
Memoria di lettura e scrittura

ROM (Read Only Memory)
Memoria a sola lettura

I/O (Input/Output)
Dispositivi di ingresso e uscita

BUS
Insieme di fili,piste o linee sui quali si muovono i segnali elettrici e che collegano un blocco funzionale con l' altro.

Per meglio comprendere come funziona ciascuno dei blocchi costituenti il microelaboratore faremo un paragone diretto con il corpo umano.
Il cervello sara' allora costituito da:

CPU + ROM + RAM

Mentre i sensi ed i movimenti del corpo possono essere assimilato al blocco di I/O (Input/Output = Ingresso/Uscita)
Avremo allora che :

la CPU
Analizza gli stimoli provenienti dall' esterno attraverso i sensi
Ricerca nella memoria ROM e RAM una risposta allo stimolo
Prende una decisione in base a cio' che esiste nella memoria
Fa eseguire l' azione

ROM
Puo' essere paragonata a tutto cio' che nella memoria umana e' il bagaglio di conoscenze che derivano dall' educazione , dall' esperienza e dallo studio.
Esse condizionano il comportamento e non possono esere modificate.
N.B. Cio' non e' vero nel caso del cervello ma lo prenderemo come assunto per semplicita di trattazione.

RAM
Nella parte RAM del nostro cervello sono depositate, cancellate e nuovamente depositate tutte quelle informazioni che e' necessario saper momentaneamente o per uno specifico scopo.
Esse non condizionano permanentemente il nostro comportamento.

I/O
Gli input possono essere paragonati ai sensi tramite i quali si recepiscono le informazioni (vista,tatto,udito, ecc.)
Gli output possono essere assimilati alle parole ed ai movimenti del nostro corpo.

BUS

I bus dei dati non sono altro che le interconnessioni nervose ovverosia i mezzi che permettono le comunicazioni all' interno del nostro corpo.

Nel nostro caso ovviamente il corpo del miscrocalcolatore.

L' esempio che riportiamo ci sembra abbastanza significativo

Sempre assimilando il funzionamento di un microelaboratore (in generale ma nel nostro caso il COMMODORE VIC 20) a quello di una persona e per meglio comprenderne l' analogia di struttura e limitatamente a certi aspetti, di comportamento consideriamo il seguente esempio.

Un autista che sta guidando la sua automobile in una citta' che non conosce chiede ad un passante dove si trovi una certa via.

Quest' ultimo gli risponde che proseguendo diritto egli dovra' voltare a destra dopo il primo semaforo che incontra.

Riportando il ragionamento a livello di istruzioni del microelaboratore la CPU richiama dalla ROM tutte le nozioni necessarie per poter guidare l' auto.

Ecco allora che quando scatta il rosso l' autista si ferma al semaforo (il suo comportamento e' stato e sara' sempre condizionato dalle informazioni depositate in ROM ogni qual volta si trovera' di fronte ad un semaforo).

L' informazione momentanea:

VOLTARE A DESTRA DOPO IL PRIMO SEMAFORO

e' stata depositata nella RAM cosi' che la CPU ne tiene conto SOLO in quel determinato caso dando il comando agli arti di eseguire la curva a destra dopo QUEL semaforo.

Da allora in poi il nostro autista utlizzera' il programma:

VOLTA A DESTRA DOPO QUEL DETERMINATO SEMAFORO

SOLO quando dovra' andare in quella certa strada.

Infatti non voltera' a destra ad ogni semaforo che incontra !!.

IN CHE MODO FUNZIONA UN MICROELABORATORE

Abbiamo detto che all' interno di un microelaboratore circolano dei segnali elettrici che passano da un blocco all' altro attraverso il BUS dei dati.
Questi particolari segnali elettrici si chiamano BIT e altro non sono che livelli alti o bassi di tensione.
Con la parola BIT contrazione delle due parole inglesi BINARY DIGIT si designa l' unita' elementare di informazione ovverosia uno dei due possibili stati o livelli logici ZERO e UNO.
Si parla allora di livello basso o stato logico ZERO 0 e di livello alto o stato logico UNO.
N.B. si usa lo zero in una forma particolare e cioe' con una sbarra diagonale per non confonderlo con la lettera O.

Se consideriamo il classico esempio di una lampadina potremo avere che questa e' in una delle due condizioni:

Accesa = ON
Spenta = OFF

Quindi con un solo filo possiamo avere 2 combinazioni di un BIT e cioe' ZERO oppure UNO, ON oppure OFF.
Se avessimo due lampadine, e quindi due fili sui quali passa corrente, ovverosia due BIT possiamo avere quattro combinazioni diverse:

0 0
1 0
0 1
1 1

Siccome nell' elaboratore i segnali si muovono su

piu' fili, ognuno dei quali puo' portare il livello logico di ZERO o di UNO. avremo cosi' che il numero di combinazioni possibili e' dato dalla seguente formula:

2 alla n = numero di combinazioni.

Dove n e' il numero dei fili.
Allora con otto fili ovvero con una serie di otto BIT e' possibile avere:

2 elevato alla 8 = 256 combinazioni diverse

Con 16 fili cioe' con 16 BIT e' possibile avere:

2 alla 16 = 65536 combinazioni diverse

Il tipo di numerazione che abbiamo introdotto e che utilizza i due stati ZERO e UNO e' detta Numerazione Binaria o:

CODICE DIGITALE BINARIO

Con un numero binario che e' costituito da una serie di BIT (o stati logici ZERO e UNO) e' possibile rappresentare un numero decimale.
Allora il numero binario di quattro bit 0110 diremo che equivale al numero decimale 6.
Per poter comunicare con il VIC dovremo allora immettere i dati o i comandi come serie di 0 e 1.
In questo caso noi comunichiamo con l' elaboratore in:

LINGUAGGIO MACCHINA

ovverosia nel linguaggio proprio della macchina che come abbiamo visto PARLA il binario.
Sarebbe pero' troppo lungo, noioso e fonte di errori, comunicare con parole fatte di ZERO e UNO

così che al posto del codice binario si può utilizzare un codice più semplice detto:

CODICE ESADECIMALE

Questo codice fà riferimento al sistema o calcolo esadecimale che si fonda su una base o radice di 16 così come il sistema decimale si fonda su una base di dieci e quello binario su base di due.
Osservare la tavola relativa alle trasformazioni fra codice decimale, esadecimale e binario.

MOVIMENTO E CONTROLLO DEI DATI IN UN MICROPROCESSORE A OTTO BIT

MICRO 6502

Il nostro VIC 20 e' basato su una CPU ovverosia su un microprocessore a 8 bit.
Cio' significa che esso e' in grado di trattare otto BIT contemporaneamente.
I dati sotto forma di parole costituite da otto BIT ovverosia da un BYTE:

OTTO BIT = UN BYTE

si muovono in ingresso ed in uscita nel BUS dati formato da otto linee.
Esiste pero' un altro BUS detto :

BUS INDIRIZZI

che nel nostro caso e' formato da sedici linee e per mezzo del quale la CPU legge nella memoria RAM o ROM o dall' INPUT/OUTPUT oppure scrive nella memoria RAM o invia all' INPUT/OUTPUT il dato.
Con un BUS indirizzi di 16 come abbiamo detto il microprocessore ha la possibilita' di indirizzare cioe' di leggere o scrivere il dato in una qualsiasi delle 65536 locazioni di memoria diverse ed ognuna identificabile appunto tramite il BUS INDIRIZZI.
Il numero dei fili cioe' dei BIT indirizzabili attraverso il Bus Indirizzi definisce infatti la capacita' di indirizzamento del sistema dove per capacita' di indirizzamento si intende la quantita' di locazioni di memoria che il microprocessore e' in grado di riconoscere.

Per locazione di memoria si intende l' insieme di otto cellette elementari (ciascuna in grado di memorizzare un BIT) in cui vengono memorizzati otto

BIT ovvero otto stati logici.
Esempio 00101000

Ad ognuna delle possibili 65536 locazioni di memoria corrisponde uno ed un solo indirizzo.
In ogni locazione di memoria che come abbiamo visto puo' contenere otto bit, cioe' un BYTE, attravreso il BUS dati si potra' depositare o leggere una delle 256 diverse combinazioni bunarie possibili con otto BIT.

IL PROGRAM COUNTER

Un programma in una definizione molto stringata, consiste in una serie di istruzioni come per esempio:

Prendi il contenuto della locazione di memoria X

Sommalo al contenuto della locazione di memoria Y

Riporta il risultato nella locazione di memoria Z

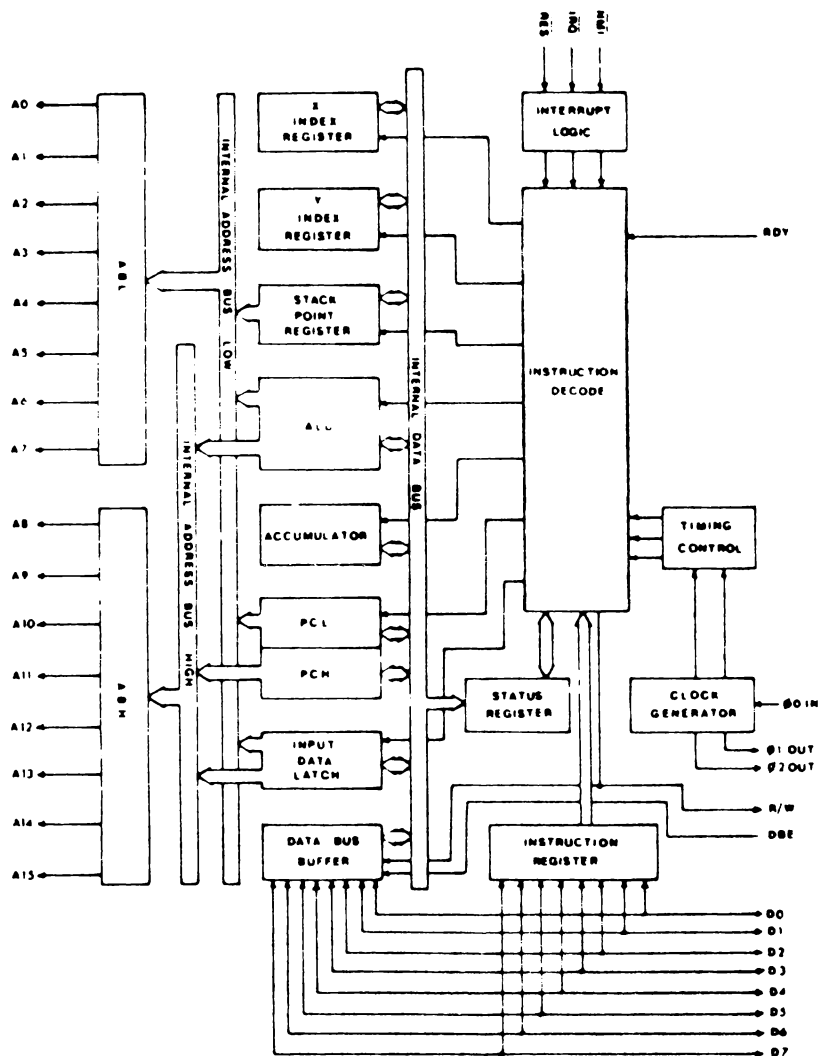
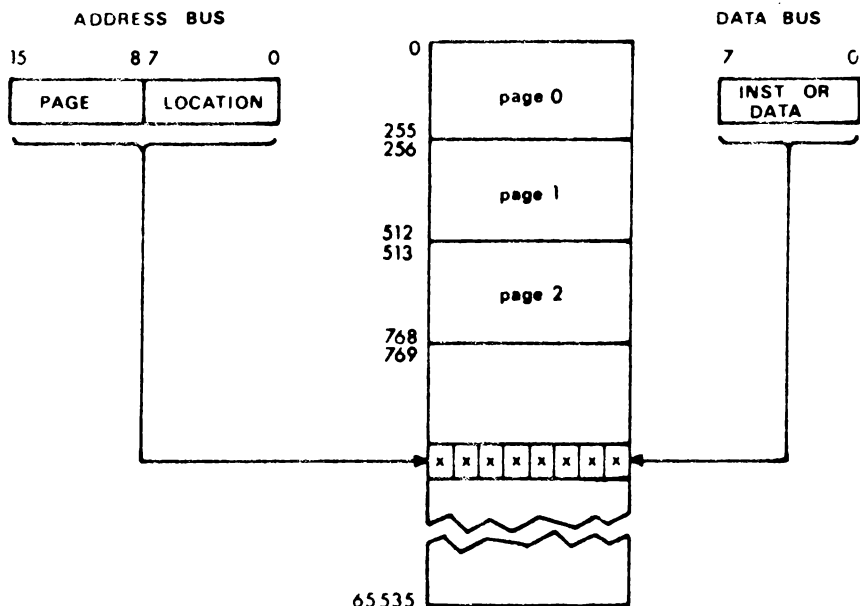
Questo come altri programmi piu' complessi puo' essere eseguito dalla CPU.
Evidentemente pero' la CPU non puo' iniziare ad eseguire istruzioni partendo da un punto qualsiasi della memoria e seguendo un' ordine casuale, ma deve iniziare l'esecuzione partendo da un punto ben preciso che rappresenta l' inizio del programma.
In concreto nella CPU si trova un registro chiamato:

PROGRAM COUNTER cioe' contatore di programma

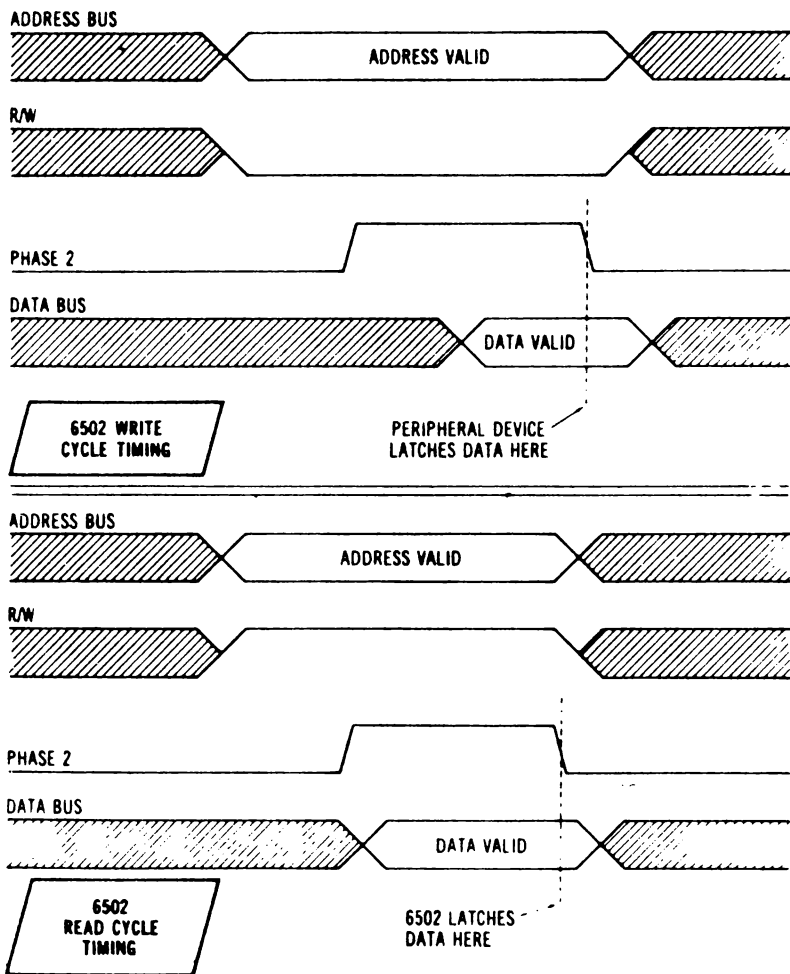
che contiene l'indirizo della locazione di memoria dalla quale verra' prelevata la prossima istruzione del programma che la CPU deve eseguire.
In linguaggio un po' piu' tecnico il PROGRAM COUNTER che si abbrevia di norma in PC, PUNTA la

prossima istruzione da eseguire.
Per questo motivo si parla anche di PUNTATORE (dall' inglese POINTER).
Dopo aver eseguito una istruzione del programma il Program Counter si posiziona automaticamente sull' indirizzo dell' istruzione da eseguire subito dopo e non richiede pertanto alcun intervento da parte dell' utente.

Le istruzioni date al microelaboratore 6502 non sono tutte di una sola locazione di memoria, ma possono interessare e lo vedremo poi, da una a tre locazioni o meglio LUNGHEZZE.
In questo caso il Program Counter si incrementera' di uno ,di due o di tre passi puntando all' istruzione successiva.

Il contenuto del PC puo' infine essere modificato dall' utente grazie ad alcune istruzioni che permettono di saltare da un punto all' altro del Programma senza dover seguire una rigida sequenza di comandi.
Il pc e' un registro da 16 BIT proprio perche' deve indirizzare parole di un contenuto di memoria e quindi gli sono necessari per l' identificazione 2 BYTE.

STRUTTURA INTERNA DEL 6502

IL CONCETTO DI IMPAGINAZIONE

Vss	1	40	$\overline{RES}$
RDY	2	39	ø2 (out)
ø1(out)	3	38	S.O
$\overline{IRQ}$	4	37	ø0 (in)
N.C	5	36	N.C
$\overline{NMI}$	6	35	N.C
SYNC	7	34	R/W
Vcc	8	33	D0
A0	9	32	D1
A1	10	31	D2
A2	11	30	D3
A3	12	29	D4
A4	13	28	D5
A5	14	27	D6
A6	15	26	D7
A7	16	25	A15
A8	17	24	A14
A9	18	23	A13
A10	19	22	A12
A11	20	21	Vss

PIEDINATURA DEL 6502

SEGNALI DI CONTROLLO TRASFERIMENTO DATI
IN UN SISTEMA 6502

M O D I   D I   I N D I R I Z Z A M E N T O

Ciascuna istruzione di un programma in linguaggio macchina contiene le informazioni sulla posizione dei dati sulla quale questa istruzione dovra' operare.
La stessa istruzione puo' essere messa sotto diverse forme in dipendenza dal punto in cui queste istruzioni sono localizzate, ognuna di queste forme e' riferita in modo particolare ad un modo di indirizzamento.
Esistono 13 diversi modi di indirizzamento e molte istruzioni possono essere date in piu' di un modo.
Per esempio l' istruzione LDA puo' essere data con ben otto modi di indirizzamento.
Questi modi possono essere divisi in 7 modi base e 6 modi accessori che sono le combinazioni di uno dei modi base insieme ad un modo di indirizzamento cosi' detto " INDEXATO".

La seguente tabella mostra tutti e 13 modi di indirizzamento:

IMPLICITO

ACCUMULATORE

IMMEDIATO

ASSOLUTO

IN PAGINA ZERO

RELATIVO

INDIRETTO

Questi altri sono i modi combinati di indirizzamento

ASSOLUTO X INDICIZZATO

ASSOLUTO Y INDICIZZATO

PAGINA ZERO X INDICIZZATO

PAGINA ZERO Y INDICIZZATO

INDICIZZATO INDIRETTO

INDIRETTO INDICIZZATO

Il piu' semplice modo di indirizzamento e' il MODO IMPLICITO, che e' usato esclusivamente per istruzioni di un singolo Byte che operino direttamente verso l' interno dei registri del microprocessore.
In una istruzione come CLC (clear Carry) non e' necessario accedere a nessun dato ragion per cui non e' richiesto nessun indirizzo.
Il modo di indirizzo Accumulatore e' usato in istruzioni che consentono operazioni logiche sui dati nell' Accumulatore.
Questo modo e' una versione particolare dell' Indirizzo Implicito e tutte le istruzioni sono di un singolo Byte.

Il modo di indirizzamento IMMEDIATO e' usato ogni volta che il programmatore desidera eseguire una operazione usando una costante. Per mettere un valore, ad esempio 25, nell'Accumulatore bisogna utilizzare l' istruzione LDA in modo Immediato
In questa forma di indirizzamento il dato e' immagazzinato nel Byte immediatamente seguente l' OPCODE (o codice operativo).
Ne' il modo di indirizzo immediato ne' il modo di indirizzamento IMPLICITO usano un indirizzo di memoria per immaggazzinare i dati e quindi sono di

uso molto modesto per operazioni con delle variabili.
Per indirizzare una qualsiasi locazione di memoria saranno necessari due Bytes di indirizzo immagazzinati nella parte operando dell' istruzione.
Questi indirizzi puntano ad una locazione di memoria dove la variabile sopra alla quale deve esere eseguita l' operazione e' collocata in quel preciso momento o dove deve essere immagazzinata.
Questa forma di indirizzamento e' nota come forma di indirizzamento assoluto.

Una forma particolare di indirizzamento assoluto puo' essere usata quando la locazione di memoria alla quale si deve accedere e' situata nella Pagina Zero della memoria.
Questo e' il solo caso in cui il concetto di pagina abbia una qualche importanza nel micro 6502.
Infatti la pagina Zero occupa le prime 256 locazioni di memoria.
Questo sistema e' chiamato :

INDIRIZZAMENTO IN PAGINA ZERO

ed usa un solo Byte di indirizzo per puntare ad una locazione di dati in pagina Zero. Questo e' possibile perche' in questo caso esiste appunto l' implicita premessa che ci si riferisce alla pagina zero.
Una istruzione di indirizzo in pagina zero quindi di due Byte e' molto piu' veloce che la stessa istruzione di indirizzo, indirizzo ovviamente in modo Assoluto, di tre Bytes, ed e' buona pratica di immagazzinare tutte le variabili in pagina Zero, cosa che ha fatto chi ha scritto il Sistema Operativo del VIC.

Quando sta girando un programma in linguaggio macchina sul VIC solo i primi 143 bytes della pagina zero dovrebbero essere usati per

immagazzinare i data ,perche' immagazzinando dati nelle locazioni superiori della pagina zero si puo' causare un blocco del sistema.

Una forma speciale di indirizzamento e' usata eclusivamente per eseguire operazioni di deviazione e di salto, ed e' conosciuta come 'INDIRIZZAMENTO RELATIVO .
In questo modo di indirizzamento la istruzione e' seguita da un operando di un singolo bytes.
Cio' non specifica un indirizzo come nel modo di indirizzamento in pagina zero, ma uno spostamento di indirizzo dove l' istruzione di deviazione e' immagazzinata. Poiche' lo spostamento puo' essere positivo o negativo l' ottavo bit e' usato per spiegare la direzione di salto cio' consente che questo sia di 127 bytes in avanti o di 128 bytes in dietro.
In alcuni programmi puo' essere necessario di avere un indirizzo calcolato invece di un indirizzo fisso come invece avverrebbe nell' indirizzamento assoluto.
Questo si puo' fare usando un modo di indirizzamento indiretto.In questo modo le istruzioni hanno un byte di indirizzo poiche' gli indirizzi dei dati non sono immagazzinati direttamente nell' operando dell' istruzione ma direttemante in pagina zero. Tutti gli accessi indiretti sono indicizzati fuorche' per le istruzioni di JMP.

L' indirizzamento indicizzato usa i contenuti di uno o di due registri indice come una devizione agli indirizzi immagazzinati nell' operando della parte dell' istruzione. L' indirizzo immagazzinato nell' operando puo' essere sia un indirizzo assoluto di due bytes oppure un indirizzo in pagina zero di un byte.
Cio' da' un totale di quattro modi di indirizzamento indicizzato, due per ogni registro

indice. L' uso principale dell' indirizzamento indicizzato e' nella possibilita' di accedere a successive locazioni di memoria usate per l' immagazzinamento di tavole o blocchi di data.

Nel modo di indirizzamento INDIRETTO INDICIZZATO, il registro indice X e' aggiunto all' indirizzo dell' operando in pagina zero che punta alle locazioni dove i sedici bit di indirizzo DATA sono immagazzinati.

Nel modo di indirizzamento INDICIZZATO INDIRETTO per prima cosa e' trovato il puntatore di indirizzo a 16 bit situato in pagina zero, successivamente deviato dal contenuto del registro indice Y che da il VERO indirizzo dei dati.
La locazione del puntatore e' fissa , mentre invece in questo modo diviene una variabile essendo deviata dal contenuto del registro indice X.

Quest' ultimo modo di indirizzamento combina i vantaggi di indirizzare qualsiasi punto della memoria con i vantaggi di deviazione forniti dai contenuti dei registri indice.

IL REGISTRO DI STATUS E L' USO DEI FLAGS.

Il registro di processo occupa una posizione molto importante nell' architettura di un sistema basato sul microprocessore 6502.
Questo e' un registro programmabile di otto bit, tuttavia a differenza degli altri registri la sua funzione e' posta fra la sezione di controllo e la sezione di registro del processore.
E' il solo registro che attui controlli logici.
Sette dei suoi otto bits o FLAGS sono usati ed ognuno ha una specifica funzione.
I flags di questo registro si dividono in tre categorie.

Alcuni controllabili solo tramite programma, altri controllabili sia tramite programma sia tramite processore e gli ultimi che vedremo sottoposti SOLO al controllo del microprocessore.
Solo uno di questi bit o flag rientra nella prima categoria ed e' esattamente il D flag o DECIMAL MODE REGISTER che occupa il bit n. 3 del REGISTRO DI STATO.
Questo flag controlla che il processore esegua le operazioni aritmetiche in binario o in decimale.
Come abbiamo visto usando l' istruzione SED, tutte le operazioni vengono eseguite in modo decimale fino a quando il Flag D non sia settato tramite un istruzione CLD o Clear Decimal Mode.

I flags che appartengono alla seconda categoria, cioe' quelli controllabili sia dal programmatore che dal microprocessore sono i seguenti:

IL CARRY

L' OVERFLOW

L' INTERRUPT DISABLE.

Il Carry o C-Flag e' localizato nel bit zero del Registro di Stato ed e' modificato sia dal risultato di determinate operazioni aritmetiche sia dal programmatore.
Il Carry e' anche usato come nono bit durante le operazioni aritmetiche o per le istruzioni di SHIFT o di ROTATE.
L' istruzione usata per settare il flag di Carry e' l' istruzione SEC, mentre per azzerare questo bit si adopera il comando CLC.
L' Overflow o Flag V occupa il Bit n. 6 del registro di stato ed e' usato durante operazioni aritmetiche con il segno per indicare che il risultato era di valore piu' grande di quanto poteva essere contenuto nel settimo bit del Byte specificato.

L' Overflow ha lo stesso significato del Flag C o bit di Carry, ma indica anche che una routine di correzione segno deve essere usata se questo bit e' in posizione di ON fino a quando l'Overflow stesso non avra' cancellato il segno nel settimo bit.
Solo il programmatore puo' riportare a zero il Flag V usando l'istruzione CLV.

Il bit di INTERRUPT DISABLE o I Flag controlla le operazioni del microprocessore durante le richieste di interrupt ed si trova nel bit n. 2 del Registro di stato.
Come vedremo in seguito gli interrupts hanno un ruolo molto importante nel funzionamento del VIC e tutte le volte che c'e' un Interrupt il processore setta l' I Flag.
L' I Flag puo' essere settato dal programmatore con l' istruzione SEI per prevenire una interruzione sul lavoro del microprocessore come durante, ad esempio, una istruzione di LOOP.
Alla fine di un programma o di una routine come quella che abbiamo detto prima la linea di Interrupt puo' essere riportata alla sua normale funzione riportando a zero il Flag I CLI.

Gli ultimi tre registri sono:

ZERO

NEGATIVE

BREAK

e sono controllati esclusivamente dal microprocessore.

Il Bit Zero o Z flag e' settato dal processore ogni qual volta il risultato di una operazione e' zero come quando per esempio si sottrae l' uno da l' altro due numeri dello stesso valore.
Il Bit Negative o N Flag e' posto usuale al settimo

bit dal processore in base al risultato di una operazione; uno degli usi piu' importanti di questo Flag e' durante operazioni aritmetiche in binario con segno.
Infatti se e' settato l'N flag il risultato sara' allora un numero negativo.
Il Bit di Break o B Flag e' settato dal processore durante una sequenza di Interrupt.
Il Flag Z occupa il bit n 1, il flag N il bit 7 ed il Flag B il bit n 4 del Registro di Stato

I sette Bit o Flags dello status register hanno ognuno un significato per il programmatore in un particolare punto del programma.
Sebbene il Carry e l' Overflow siano presenti nelle operazioni aritmetiche il maggior uso dei Flags e' in combinazione con le operazioni di salto condizionato.
Cio' da al programmatore la possibilita' di incorporare gruppi di istruzioni decisionali entro un programma.
Per testare uno di questi bit e' necessario eseguire un 'azione come quelle che riportiamo.
Un salto condizionato ha la stessa funzione del IF.....THEN.....GOTO del Basic. per cui ci sono un gruppo di queste istruzioni che eseguono diverse funzioni e testano i differenti Flags.

LE DEVIAZIONI, I SALTI ED IL PROGRAM COUNTER

Per comprendere l' uso delle istruzioni di deviazione e di salto e' necessario prima spiegare il concetto di sequenza nel programma e il suo controllo tramite un contatore dei passi di programma o PROGRAM COUNTER.
In pratica si tratta di definire il concetto di subroutine.
Il contatore di programma o PROGRAM COUNTER consiste di due registri di otto bit.
Come gli altri registri, questi comunicano con il processore attraverso il Data Bus ma l' uscita e' anche connessa a 16 linee del bus indirizzi.
Uno dei registri del PROGRAM COUNTER e' connesso alle linee di indirizzo (le otto piu' alte) ed e' chiamato PROGRAM COUNTERL, mentre l' altro e' connesso alle otto linee piu' basse e si chiama PROGRAM COUNTER.
Sebbene si tratti di due registri da otto bit ciascuno il funzionamento del PROGRAM COUNTER e' perfettamente identico ad un registro da sedici bit.
E' il PROGRAM COUNTER che controllera' l' indirizzamento della memoria o i puntatori dell' indirizzo dato perche' contiene l' indirizzo della successiva locazione di memoria che deve essere esaminata dal processore.
All' inizio del programma il PROGRAM COUNTER conterra' l' indirizzo della prima istruzione.
Questa e' una delle funzioni dell' OPERATING SISTEM RESET SOFTWARE ed e' anche eseguita dai comandi SYS e USR quando si passa da un programma BASIC ad un programma in Linguaggio Macchina.
L' istruzione caricata dalla memoria e' immgazzinata in un registro di istruzioni per essere decodificata e quindi resa COMPRENSIBILE, dall' unita' di controllo.

Questo processo richiede un ciclo di CLOCK (per vedere che cosa sia il Clock e come operi vedere il capitolo relativo) , durante questo periodo il Program Counter e' incrementato di UNO e punta quindi alla successiva locazione di memoria.
Di norma il processore richiede piu' di un Byte per interpretare un' istruzione.Il primo Byte contiene l' operazione base ed e' conosciuto come OPERATION CODE o OP CODE mentre i successivi Bytes conosciuti come OPERANDS possono contenere sia un Byte di dati o l' indirizzo al quale sara' necessario accedere per questi dati.
Un' istruzione puo' richiedere fino a tre locazioni di memoria sequenziali per il PROGRAM COUNTER per primo punta all' OPERATION CODE che e' andato a prelevare dalla memoria e lo immagazzina nel registro di istruzione.
Il PROGRAM COUNTER e' quindi incrementato e punta alla successiva locazione di memoria i contenuti della quale sono riportati e immagazzinati nella ALU (Aritmetic Logic Unit).

Dopo aver completato l' operazione che usualmente richiede circa quattro cicli di Clock. il processore incrementa il PROGRAM COUNTER per puntare alla successiva istruzione e il processo si ripete.
In questo modo il PROGRAM COUNTER continuera' ad avanzare fino alla locazione massima di memoria riportando istruzioni ed indirizzi, o fino ad uno stop.

LO STACK REGISTER ED SUO USO

Lo Stack Register e' principalmente usato per la manovra degli Interrupts e delle Subroutines.
E' un registro di otto bits la cui funzione e' identica a quella del PROGRAM COUNTER.
E' usata per puntare ad un indirizzo in pagina UNO della memoria (locazionei decimali da 256 a 511) conosciuta come AREA DI STACK .
Lo Stack e' settato quindi dalle locazioni di memoria che iniziano dal decimale 511 indietro fino ad un massimo di 255 Bytes. E' un registro che viene utilizzato con il metodo LIFO cioe' Last In First Out.
Cio' consente che l' ultimo bit immagazzinato nello Stack sia il primo al quale si ha accesso.
Tutte le volte che un dato e' inserito nel' area di Stack lo Stack Pointer e' decrementato di uno, mentre invece quando ne esce lo Stack Pointer e' incrementato di uno.
Il metodo di indirizzamento dello Stack e' indipendente dal programma ed e' basato unicamente su una serie di eventi cronologici.
Lo Stack in effetti e' usato come una zona di immagazzinamento temporaneo.
Tutte le volte infatti che e' chiamata una Subroutine in un programma in Linguaggio Macchina l' attuale contenuto del Program Counter deve essere salvato.
Al ritorno dalla subroutine il programma deve ripartire dalla corretta locazione. Nello stesso modo tutte le volte che e' interrotto il funzionamento del microprocessore, l' indirizzo corrente e' salvato nel PROGRAM COUNTER prima che il microprocessore faccia eseguire la routine di Interrupt.
Una Subroutine puo' richiamare altre Subroutines che richiedono quindi l' immagazzinamento di piu'

indirizzi nello Stack.
L' ultimo indirizzo di rientro immagazzinato sara' il primo indirizzo richiamato nel Program counter alla fine della Subroutine, di qui la necessita di struttura LIFO dello Stack Pointer.
Il sistema di chiamare delle Subroutines tramite altre Subroutines e' chiamato " SUBROUTINES NESTING ed e' molto comune nei programmi in Linguaggio Macchina.
L' organizzazione dello Stack nel 6502 limita l' utente ad un massimo di 127 livelli di Nesting che e' in effetti una cifra maggiore di quanto normalmente si possa usare.
Le Subroutines in Basic usano lo Stack per immagazzinare il ritorno di indirizzi ed i contenuti dei registri.

Una subroutine e' richiamata tramite una istruzione JSR o Jump To Subroutine cioe' Salta alla Subroutine.
Questo comando spinge il contenuto corrente contenu del PC entro lo Stack.
Una locazione di memoria immagazzinata come OPERAND FIELD viene quindi immgazzinata entro il PC. Cio' induce il Micro a saltare ad una nuova sub del programma ed a iniziare l' esecuzione da una locazione immagazzinata nel PC.
Il rientro da una subroutine al programma principale e' eseguito attraverso una istruzione RTS o Return From Subroutine cioe' ritorno da Subroutine.
L' esecuzione di questo comando carica l' indirizzo di ritorno dallo Stack Pointer entro il Program Counter ed incrementa il PC per puntare all' istruzione seguente il comando JSR.
Anche lo S P e' incrementato per puntare al successivo indirizzo di subroutine se ne esiste uno.
Lo Stack puo' essere usato dal programmatore come una locazione di immagazzinamento temporaneo per i

dati passati alla subroutine.
Al programmatore infatti necessitano un gruppo di istruzioni che gli consentano di mettere questi dati entro lo S e poi di rileggerli.
Il contenuto corrente dell' Accumulatore puo' essere trasferito alla successiva locazione di memoria nello Stack Register attraverso un comando di PHA o Push Accumulator On To Stack.
I dati possono essere letti dalla corrente locazione puntata attraverso lo Stack Pointer entro l' Accumulatore attraverso il comando PLA o Pull Accumulator From Stack.
Tutte queste istruzioni provvedono che lo Stack Pointer sia automaticamente incrementato o decrementato di uno.
Un esempio di immagazzinamento dati nello Stack e' di poter salvare il contenuto dello Status Processor e dei registri indice quando viene chiamata una subroutine.
I contenuti dello status register possono essere inseriti entro lo Stack tramite i comandi PHP o Push Processor Status Onto Stack.
Succesivamente possono essere ritrasferiti dallo Stack allo Satus Register attraverso il comando PLP o Pull Processor Status From Stack.
Per salvare il contenuto dei registri indice bisogna trasferire all' accumulato re il contenuto di questi registri e dopo inserirli nello Stack.
Quando si sta scrivendo una routine in Linguaggio Macchina per il VIC che sara' successivamente richiamata da un programma Basic e' molto importante di salvare prima di tutto il contenuti del Processor Accumulator e dei registri indice nello Stack.
I contenuti di questi registri infatti saranno succesivamente rimessi a punto prima di tornare al Basic in quanto se cio' non avvenisse potrebbe causare danni al programma.
Normalmente lo S P punta ad una locazione in pagina uno e questa locazione sara' automaticamente incrementata o decrementata da ordini del

processore.
In alcuni casi il programmatore ha necessita' di cambiare il contenuto dello Stack Pointer.
Allora lo Stack Pointer e' caricato tramite il trasferimento dei contenuti del Registro indice X (operazione che avviene tramite l' istruzione TXS o Transfer Index X To Stack Pointer).
Questa istruzione e' usata all' inizio di un Programma per inizializzare lo Stack Pointer ed e' eseguita automaticamente sul VIC come parte della routine di POWER UP.
La preinizializzazione dello Stack sul VIC puo' causare dei problemi allora sara' necessario rileggere il corrente contenuto dello Stack Pointer e cercarlo nuovamente sul Registro indice X con l' istruzione TSX o Trasfer Stack Pointer to Index X

I REGISTRI INDICE

L' avere un indirizzo fisso in un campo operativo di una istruzione pone dei problemi quando si debba accedere ad un blocco di dati sequenziali come potrebbe essere una tavola o un buffer in posizione di Input.
Uno potrebbe essere di usare una stringa per caricare le istruzioni in una certa forma:
Carica i dati dall' indirizzo 1
Esegui l' operazione
Carica i dati dall' indirizzo 2
Esegui l' operazione

e cosi' via.
Questo occupa una larga parte di spazio della memoria e oltre tutto a discapito della efficienza e della rapidita' di esecuzione del programma.
Un approccio piu' sofisticato e' nell' uso di contatori il contenuto dei quali sia automaticamente aggiunto all'indirizzo del campo operando dell' istruzione.

Due contatori chiamati anche registri indice sono presenti nel 6502 ambedue di otto bit:

REGISTRO INDICE X

REGISTRO INDICE Y

Questi sono usati tramite istruzioni in uno dei modi di indirizzo indicizzati.
Il piu'semplice e' il modo di indirizzamento indicizzato assoluto nel quale il contenuto di un registro indice e' aggiunto all' indirizzo del campo operando dell' istruzione, per cui si ottiene un nuovo indirizzo al quale si potra accedere per i DATA.
Il fatto che un registri indice sia di solo otto bit limita la dimensione massima del blocco data accessibile, usando l' indirizzo indicizzato, a 256.
In pratica pero' la maggior parte delle tavole e' piu' corta per cui non e' una significativa limitazione.
Se e' necessario manipolare tavole piu' lunghe allora le tecniche di programmazione come l' indirizzo indiretto indicizzato serviranno a superare questa limitazione.
I registri indice sono controllati e manovrati da un gruppo di istruzioni particolari.
Un numero puo' essere caricato dentro o immagazzinato o richiamato dal registro indice tramite le istruzioni LDX LDY STX. STY.
Nello stesso modo il contenuto dei registri indice puo' essere confrontato con valori in memoria per vedere se e' necessario un salto condizionato ad una determinata posizione usando le istruzioni CPX e CPY.
Il contenuto di un qualsiasi registro indice e' modificato per puntare alla succsiva locazione incrementando lo stesso registro di uno o decrementandolo.
Per incrementare questi registri useremo le

istruzioni INX o INY e per decrementarli DEX o DEY.
Le rimanenti istruzioni relative ai registri indice consentono il trasferimento dei contenuti dell' Acccumulatore in uno dei registri indice e Viceversa.
Si tratta ddelle istruzioni TAX e TAY che trasferiscono il contenuto dell' Accumulatore dentro i registri X e Y oppure delle istruzioni TXA e TYA che trasferiscono i contenuti dei registri indice nell Accumulatore.
In alcuni programmi si rende necessario avere uno indirizzo calcolato piuttosto che avere un indirizzo di base con una deviazione, come nel modo di Indirizzo Indicizzato Assoluto
Cio' e' eseguito usando l' indirizzamento indiretto in cui le istruzioni hanno appena un Byte di Campo indirizzi che puntera' all' indirizzo effettivo come due Bytes in pagina Zero.
L' indirizzo dati non sara' allora immagazzinato direttamente nel campo operandi dell' istruzione ma indirettamente in pagina zero e tutti gli indirizzi di accesso saranno indicizzati eccetto che per gli indirizzi di Jump o di salto.
Ci sono due modi disponibili di indirizzamento indiretto:

INDIRIZZAMENTO INDIRETTO INDICIZZATO

INDIRIZZAMENTO INDICIZZATO INDIRETTO

Il primo metodo indica che il registro X debba essere aggiunto all' indirizzo dell' operando in pagina zero.
Questo punta alla locazione dove l' indirizzo dei sedici dati e' immagazzinato.
Uno dei piu' larghi usi di questo modo di indirizzamento e' di ritrovare dei dati da una tabella o da una lista di indirizzi.
Il secondo metodo di indirizzamento funziona in questo modo.
Per primo vengono letti i sedici bit di indirizzo

in pagina zero successivamente l' indirizzo e' deviato dal contenuto del registro indice Y per dare il vero indirizzo dei dati.
La locazione del puntatore e' fissa tuttavia in questo modo diventa variabile a causa della deviazione effettuata tramite il contenuto del registro X.
Questo modo di indirizzamento combina il vantaggi di un indirizzo che puo' puntare da qualsiasi parte della memoria con la capacita' di deviazione fornita dai registri indice.
E' un metodo particolarmente potente per accedere all' ennesimo elemento di una tavola il cui indirizzo di partenza sia immagazzinato in pagina Zero.

I L C O D I C E M A C C H I N A N E L V I C

Il VIC ha un grande vantaggio sulla maggior parte dei microcomputer esistenti sul mercato in quanto puo' essere programmato sia in BASIC sia in Linguaggio Macchina. Cio' da al programmatore la potente opzione di usare Subroutines in Linguaggio Macchina entro un programma Basic.
Il VIC normalmente GIRA in Basic che, come su tutti prodotti CBM, e' residente in FIRMWARE e ci sono diverse strade di accedere al LM.
Le prime due usano comandi in Basic che sono USR e SYS. Ambedue questi comandi accedono a delle subroutines il cui indirizzo e' specificato nel comando o in una locazione particoare di pagina Zero.
Altri metodi involvono l' aggiunta di subroutines in LM entro il Sistema Operativo.
Vediamo uno per uno questi sistemi.

USR
Il comando Basic USR(X) trasferisce il controllo dei programmi ad un indirizzo immagazzinato nelle locazioni 1 e 2 della pagina Zero.
Questo indirizzo e' definibile dall' utente e sara' l' inizio della Subroutine in LM.
Il valore X specificato nel comando e' un parametro da usarsi attraverso la Subroutine che valuta ed inserisce nel Floating Accumulator n. 1 che inizia dalla locazione n. 61 (decimale) di pagina Zero

SYS
Il comando Basic SYS(X) fa saltare il controllo del programma ad una Subroutine in LM che ha l' indirizzo specificato nella X.
La X del comando SYS cioe' il suo parametro, puo' essere sia una variabile che una costante ed uguale comunque ad un indirizo decimale compreso entro la capacita' di indirizzamento del processore.

I parametri possono esser passati tra il programma Basic e la Routine in LM usando i comandi PEEK e POKE per immettere o leggere valori da una specifica locazione di memoria.
Se la routine in LM e' localizzata nella memoria ROM ed inizia dall' indirizzo esadecimale A000 allora il Sistema Operativo del Vic salta a questa locazione escludendo l' interprete Basic fino a quando la macchina resta accesa.
Questo e' molto utile perche' consente all' utente di cambiare in pratica il S. O. del VIC..
Cio' puo' essere fatto sia aggiungendo comandi Extra al Basic che cambiando le operazioni di Input e Output per mezzo dell' uso di vettori di salto situati in ROM e in RAM come accade per esempio per i giochi su Cartridge.

AGGIUNTA DI UN PROGRAMMA ENTRO LE R. DI INTERRUPT.

E' possibile aggiungere un programma entro le routines di Interrupts che vengono richiamate 16 volte al secondo attraverso lo Scanning della Tastiera.
Questo metodo per esempio consente lo scanning delle porte di I/O per un Input o la disabilitazione selettiva di certi tasti della tastiera.
Ogni situazione nella quale un programma debba girare in CONCORRENZA con il programma principale, dovrebbe usare questo metodo.

Un altro metodo consente di inserire codici extra entro la routine di CHARGOT che prende ciascuna linea di programma Basic della memoria prima che sia eseguita tramite l' interprete Basic.
Intercettando ciascuna linea di Basic prima della sua esecuzione e' chiaro che puo' essere aggiunta una nuova istruzione in Basic.
L' istruzione sara' eseguita dall'utente scrivendo una subroutine in LM.

Ambedue i metodi di inserimento dei codici nelle routine di Interrupt e in aggiunta alla Routine di CHARGOT debbono essere usati con estrema cautela.

TAVOLA DEI VETTORI DI INDIRIZZO

La tavola dei vettori di indirizzo della RAM puo' essere usata per inserire codici o per rimpiazzarli entro il Sistema Operativo o entro l' interprete Basic.

La principale ragione per l' uso di Subroutines in LM e' che il Basic e' troppo lento per alcuni lavori in modo particolare quando debbano essere usate le porte di I/O oppure per particolari esigenze di rappresentazione sullo schermo.
Una routine in LM infatti e' almeno 100 volte piu' veloce dello stesso programma scritto in Basic.
Un' altra ragione per usare il LM puo' essere, come abbiamo gia' detto per cambiare il S.O. o parte di esso oppure per un' uso particolare di alcune Subroutines dello stesso Sistema Operativo.
Un programma in LM che debba essere caricato nella zona RAM della memoria viene normalmente immagazzinato al punto massimo della memoria.
Questa area e' usata per immagazzinare le stringhe di caratteri e, per evitare sovrascritture fra le stringhe di caratteri ed il programma in LM, e' necessario cambiare i puntatori della memoria.
I puntatori della memoria sono settati al momento dell' accensione attraverso una Routine di POWER UP
Questa routine e' particolarmente necessaria nel VIC in quanto questo sistema puo' avere numerose aggiunte di memoria. Infatti la locazione massima della memoria e pertanto il valore ivi immagazzinato tramite i puntatori, dipende dalle espansioni che siano state eventualmente collegate al VIC.
Abbassando i valori di questi puntatori un blocco

della memoria puo' evidentemente essere esclusivamente riservato per l' uso di programmi in LM.
Il S.O. guardera' allora un nuovo massimo della memoria attraverso i suoi puntatori e solamente da questo punto all' indietro scrivera' le stringhe.
Il puntatore della memoria e' immagazzinato come Byte di ordine piu' basso all' indirizzo decimale 51 e come Byte di ordine piu' alto all' indirizzo 52.
le locazioni di indirizzo decimale 55 e 56 contengono i valori dei puntatori del massimo delle stringhe e sono settati al massimo della memoria, cioe' resi uguali al massimo della memoria all' atto dell' accensione della macchina.
Per questo motivo sara' necessario spostare anche questi valori.
Dei due metodi che abbiamo visto quando si debba trasferire il controllo del sistema al Linguaggio Macchina tramite il Basic e' preferibile quello del comando SYS che trasferisce totalmente il controllo dal Basic al programma o alla Subroutine in LM.
Alla fine si puo' rientrare in ambiente Basic per mezzo di un normale RITORNO DA SUBROUTINE con comando RTS inserito al termine del programma in LM.
L' istruzione SYS (X) puo' essere usata per trasferire il controllo a qualsiasi altro programma quale per esempio un programma di MONITOR o un programma di aiuto come ne sono presenti suula linea CBM (COMMAND-O,SUPER-KRAM,ecc).
Infatti se viene incontrata la seguente istruzione

20SYS (35800)

alla linea 20 il Basic cede il controllo del computer al programma che sia collocato nella posizione di memoria di partenza 35800. Per cui la forma generale dell' istruzione SYS e':

SYS (indirizzo di partenza).

Nota
L' indirizzo di partenza puo' essere anche un valore calcolato e comunque deve dare luogo ad un numero positivo non piu' grande di 65535. Ricordiamo inoltre che il comando SYS puo' esser usato anche in forma diretta.
E' bene tener presente che durante l' esecuzione di programmi in LM il Sistema Operativo del VIC non tiene conto di tutte quelle protezioni che il Basic possiede per continuare a funzionare anche qualora l' utente commetta un errore. Un esempio di quanto detto e' che quando si lavora in LM puo' capitare di entrare in un LOOP INFINITO cioe' di non riuscire a venir fuori da una routine che continua a girare perche' non trova una istruzione adeguata. Cosa che invece capita raramente in Basic e che comunque puo' essere interrotto con un RUN-STOP.
Da questo punto di vista quindi ogni volta che il controllo viene trasferito dal Basic al linguaggio macchina si puo' avere una situazione in cui il funzionamento di questi tipi di programmi causino uno stop al funzionamento della macchina stessa con la conseguenza che per uscire dovremo spengere il VIC e perdere quanto immagazzinato.
Quando non si debbano sviluppare dei programmi che non siano delle piccolissime Subroutine e' bene usare un Monitor.
Un monitor consente che un programma in LM sia scritto direttamente in memoria usando codici esadecimali e inoltre che questi programmi siano caricati e quindi salvati su nastro nel formato del LM.
Per facilitare la funzione di scrittura di un programma in LM, la COMMODORE ha messo a punto un "MACHINE CODE MONITOR" in cui e' incluso anche un semplice Assembler e Disassembler.

LA FUNZIONE USR

Esistono parecchi programmi, in particolare routine

matematiche per i quali e' piu' conveniente passare parametri da e per il Basic usando la funzione USR in modo da ottenere risultati direttamente eseguibili dal Basic.
L istruzione USR come abbiamo visto, viene specificata con un parametro.
Il Basic calcolera' l'espressione per i suoi parametri e collochera' il risultato del calcolo in un accumulatore in virgola mobile che l' interprete usa per tutte le sue funzioni.
E' da notare che se nessun parametro viene trasferito, l'accumulatore in virgola mobile, non e' inizializzato dall' utente o tramite qualsiasi altra tecnica, mentre prima dell' esecuzione della funzione USR e' stato usato dal BASIC in un gran numero di modi.
USR richiama una rutine che esegue un programma in LM.
Poiche' USR e' una funzione, e' possibile includerla come parte di una istruzione Basic come ad esempio:

If USR(A) = 1 then...

In questo caso il parametro A verra' passato alla funzione USR in Accumulatore in virgola mobile.
Il risultato dell' accumulazione in virgola mobile, quando l' utente ritorna al Basic verra' comparato a 1 e verra' eseguita la funzione logica richiesta.

Il comando SYS e' il modo piu' usuale per trasferire il controllo ad una procedura in LM in cui le variabili del programma principale non vengano usate.
USR e' il modo piu' tipico quando invece si vuole implementare un nuovo comando Basic.
Vi e' una importante considerazione da fare quando si usi USR.
Infatti il comando USR adopera delle locazioni per delle variabili preassegnate ed esattamente le locazioni decimali n1 e n .2 in pagina Zero.

Queste locazioni devono essere inizializzate con il valore esadecimale dell' indirizzo di partenza in cui il programma in LM e' stato memorizzato.
Cio' puo' essere fatto in qualsiasi momento nel programma principale attraverso l' istruzione di POKE caricando l' equivalente decimale e la parte meno significativa dell' indirizzo nella locazione 2 e la parte piu' significativa dell' indirizzo nella locazione n.1 come ad esempio:

```
10 POKE 1,122
20 POKE 2.2
30 IF USR(A) = 1 THEN....
ecc.
```

INIZIALIZZAZIONE DEL SISTEMA

Quando il VIC viene acceso e' eseguita una predefinita sequenza di inizializzazione.
Questa inizializzazione consente che il sistema in tutte le sue variabili ed in tutti i suoi vettori su RAM siano correttamente fissati.
Lo schermo, le linee di Input/Output, e la tastiera correttamente definita, la memoria contrllata e l' interprete BASIC messo in condizioni di operare in modo diretto e di accettare gli INPUT.
La sequenza di inizializzazione e' resa possibile da un circuito che si chiama :

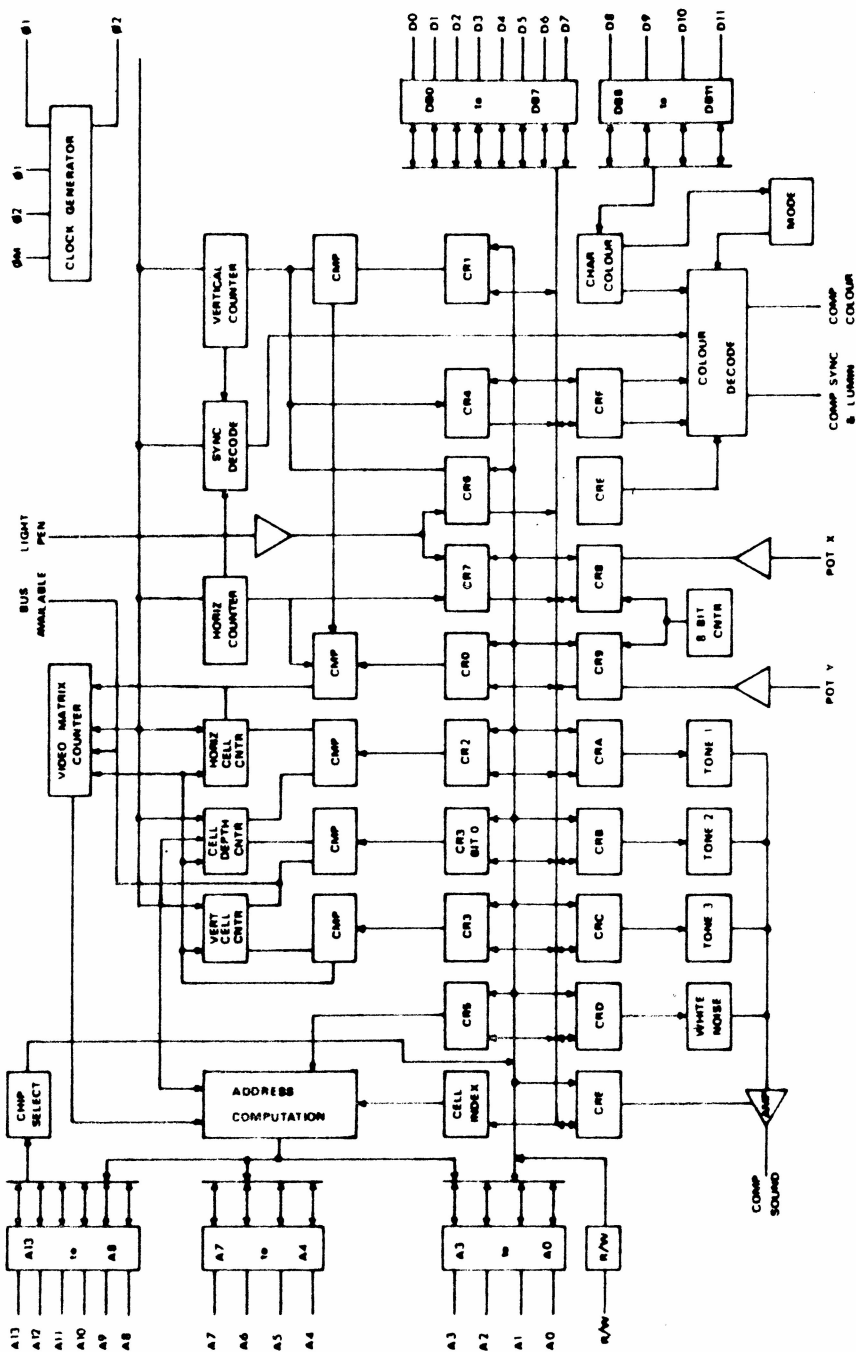
CIRCUITO DI RESET

Il circuito di reset sul VIC consiste in un integrato temporizzatore "555"
In questo modo, quando viene acceso, la linea di RESET e' mantenuta bassa per un certo periodo.
Quando la linea di RESET del Processore e' messa momentaneamente bassa (minimo 6 cicli di Clock) viene informato il processore di iniziare l' esecuzione di un programma il cui indirizzo di start e' memorizzato nelle locazioni esadecimali $FFFc e $FFFD.
La routine di partenza data da questo indirizzo e' localizzata alla locazione esadecimale $FD22.
L' area di memoria resa disponibile per le espansioni di memoria sul vic puo' essre divisa in tre sezioni.
Spazi di memoria riservato esclusavamente per memoria di tipo ROM.
Spazio riservato per sia ROM che RAM e quello riservato esclusivamente per memoria RAM.
La sezione di espansione della memoria che ci interessa in questo momento e che e' connessa con il sistema di inizializzazione, e' quella riservata all' uso esclusivo della memoria ROM che va dalle locazioni esadecimali $A000 a $BFFF
Infatti la prima funzione della routine di Start a

$FD22 e' di controllare se esiste una ROM inserita nello spazio di indirizzo $A000.
Quetsa funzione viene messa in opera testando una stringa di 5 caratteri che inizia ad una specifica locazione su ram.
La sequenza dei 5 Bytes ricercati per questo e' la seguente:

INDIRIZZO	contenut	carat. ASCII
$A004	$41	A
$A005	$30	0
$A006	$C3	rvs C
$A007	$C2	rvs B
$A008	$CD	rvs M

n.b. rvs sta per REVERSE

Se la routine di Start trova questa stringa di caratteri allora il controllo del programma salta ad un indirizzo immagazzinato nei primi due Bytes di ROM $A000 e $A001 scritti dall' utente come HARD START routine di inizializzazione.
Un secondo indirizzo di salto e' immagazzinato nelle locazioni $A002 e $A003
Questa e' una routine di WARM START che riporta il controllo del programma al BASIC ed e' richiamata quando viene premuto il tasto di RESTORE.
Se la stringa di caratteri AOCBM non e' trovata allora viene settato il FLAG 0 ed e' chiamata la routine di inizializzazione localizzata a $FD2F.
Questa particolarita' del VIC consente che il calcolatore possa essere usato in un largo spettro di applicazioni dove al programma e' richiesto che esso giri automaticamente all' accensione.

Struttura interna del 6561

IL 6561 VIDEO INTERFACE CHIP

Moltissime delle particolare funzioni del VIC sono da attribuire ad un singolo circuito integrato progettato e realizzato interamente in casa COMMODORE attraverso la MOS TECN. consociata del gruppo.
Questo integrato contiene le funzioni necessarie per la gestione e la programmazione dei colori e dei caratteri grafici compatibili con uno schermo ad alta risoluzione.
Il 6561 incorpora anche il generatore di effetti sonori, provvede alla conversione da analogico a digitale per i joysticks, consente l' uso della penna luminosa.
Tutte quste funzioni possono essere direttamente usate dal programmatore attraverso l' uso dei sedici registri di controllo del 6561 che vedremo in dettaglio.
In sintesi il 6561 VIDEO INTERFACE CHIP ha le seguenti tre distinte funzioni:

CONTROLLO DEL DISPLAY ATTRAVERSO MONITOR O TV

GENERAZIONE DEL MASTER OSCILLATOR CLOCK

CONTROLLO DELLE FUNZIONI DI INPUT/OUTPUT PER I VIDEO-GAMES

Solo la prima e la terza funzione hanno interesse per il programmatore in quanto il MASTER OSCILLATOR CLOCK e' un funzione hardware per sincronizzare il sistema con il 6561.
Il controllo e la generazione dei colori e' una delle principali funzioni del VIC.
Per ottenere questo e' possibile accedere a quattro differenti aree di memoria la locazione delle quali e le cui rispettive funzioni sono sotto controllo del programmatore.
Queste aree sono:

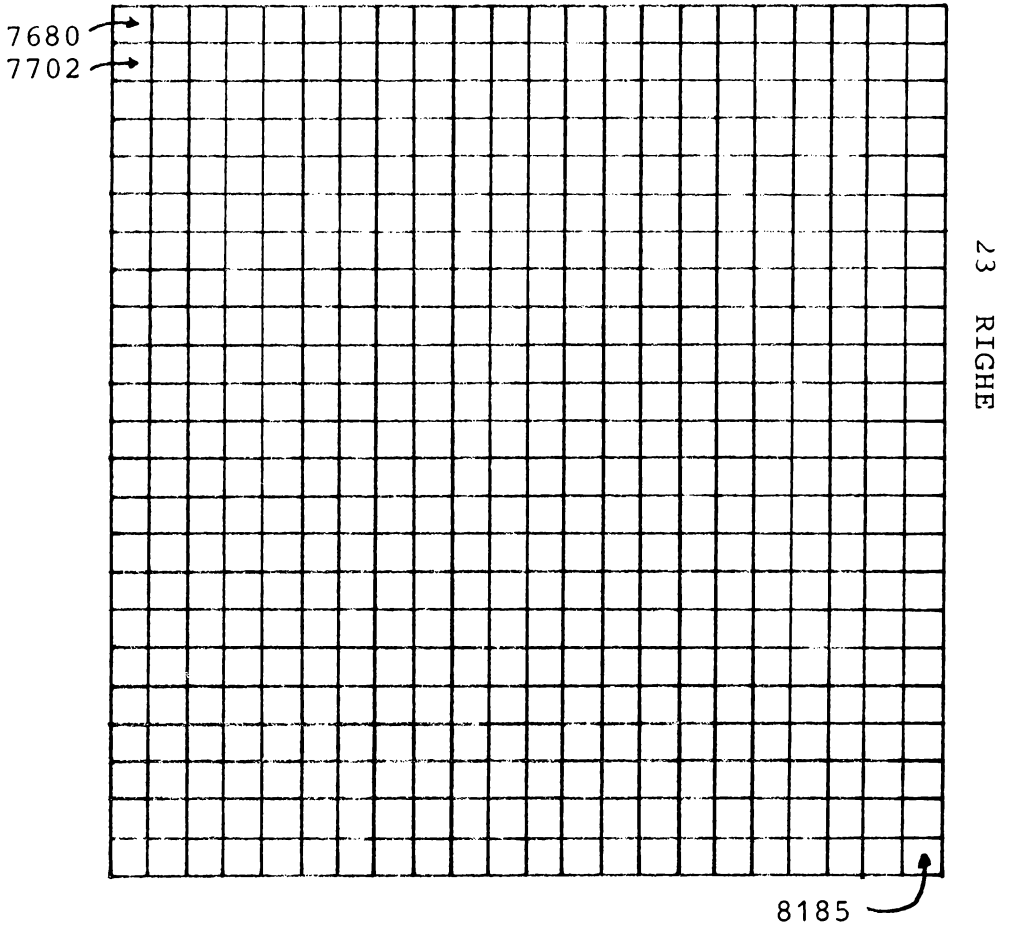
1 - VIDEO RAM CHARACTER POINTER
Ogni locazione di memoria corrisponde ad una posizione dello schermo. La locazione 0 di questa sezione della memoria RAM contiene il codice ASCII del primo carattere in alto a sinistra dello schermo, la locazione 1 del secondo, cioe' alla destra del primo e non sotto e cosi' via per tutto lo schermo.
Su di un VIC in configurazione base, quindi senza espansione, questa parte di memoria inizia dalla locazione $1e00 e prosegue per i 506 bytes necessari, mentre in presenza di una qualsiasi espansione questa area come abbiamo visto si sposta

2 COLOUR POINTER
Questa parte della memoria RAM e' di lunghezza identica alla precedente e simile nel funzionamento. Infatti contiene invece del codice ASCII, per lo stesso carattere in alto a sinistra, il codice di colore diretto e di sfondo del carattere.Su di un VIC in configurazione base questa parte di memoria inizia alla locazione esa $9600 per i soliti 506 bytes.
In presenza di espansione superiore a 3 k, l' inizio viene automaticamente spostato alla locazione esa $9800 sempre per 506 Bytes.

3 - GENERATORE DI CARATTERI
Vedremo in maniera piu' estesa nel capitolo

relativo il funzionamento con degli esempi di questa parte di memoria, ricordando per ora che il generatore di caratteri e' immagazzinato in un blocco di 4k Bytes di memoria ROM nel funzionamento normale.
Questo blocco inizia alla locazione esa $8000 e per essere variato necessita di una redifinizione in memoria RAM che normalmente deve essere aggiunta tramite il connettore esterno.

4 REGISTRI DI CONTROLLO DEL 6561
Come accennato questi registri controllano il modo di operare del 6561 e sono collocati in sedici locazioni di memoria RAM consecutive a partire da $9000 a $900f Il loro indirizzo e definito via Hardware.
All' accensione del VIC le routines di inizializzazione del sistema pongono i registri anzidetti in uno schema standard di funzionamento come segue:

Formato del video 23 righe per 22 colonne.
Uso del generatore di caratteri standard con il contenuto delle locazioni da $8000
Caratteri bleu su schermo bianco con bordo cyan.

MAPPA DELLA MEMORIA DI SCHERMO

I MODI DI VISUALIZZAZIONE DEL 6561

Il Vic ha due modi di visualizzazione (DISPLAY):
MODO NORMALE di testo.

CARATTERI DEFINIBILI dall' utente.

Questi modi sono determinati dalla posizione occupata in memoria dal GENERATORE DI CARATTERI

Ci sono anche due modi di operare sui colori:
ALTA RISOLUZIONE

MULTICOLORE

I due modi di visualizzazione dipendono dal fatto che si usi il genereatore di caratteri standard su ROM oppure si usi il generatore di caratteri programmabile su RAM.
La posizione del generatore di caratteri entro lo spazio di memoria e' determinata dal contenuto dei bits da 0 a 3 del registro di controllo in 36869
Il normale contenuto di questi 4 bits e' 0 il che da come indirizzo di partenza del generatore di caratteri la locazione esadecimale $8000 - decimale 32768.

Il generatore di caratteri standard su ROM per una estensione di 4K contiene il modello, la matrice a punti di ciascuno dei 256 caratteri che si possono rappresentare.
Questa area di memoria puo' essere a sua volta divisa in due sottosezioni di 2 K ciascuna .
La prima inizia da 32768 e va fino a 34815 e contiene la matrice a punti di 128 caratteri maiuscoli e caratteri grafici piu' la versione in REVERSE degli stessi.
La seconda sezione di memoria con start a 34816 e' identica alla prima solo che la parte grafica e'

sostituita dalle minuscole.
Ci sono due modi per accedere alla seconda parte del generatore di caratteri.
La prima e' premendo il tasto SHIFT ed il tasto COMMODORE.
La seconda strada e' data cambiando il contenuto del registro di controllo 36869 del 6561 come segue

POKE 36869,242 mette in funzione la seconda parte

POKE 36869 240 mette in funzione la prima parte del generatore.

(questo in configurazione base)

L' indirizzo di partenza del generatore di caratteri e' come abbiamo detto nel registro di controllo 36869 e quindi puo' essere spostato per definire una zona RAM in cui creare i caratteri che si desiderano.

Diamo qui una serie di regole per manovrare questo interessante aspetto della memoria del VIC.

1 - L' indirizzo di partenza del generatore definibile puo' essere un qualsiasi blocco di RAM di 2K (4K se e' usato un 8*16) fra gli indirizzi esa $1000 e $3000.

2 Deve essere collocato nella posizione piu' alta possibile di memoria.

3 Deve essere protetto da possibili sovrascritture del Basic spostando in basso i puntatori del massimo della memoria.

4 - L' indirizzo iniziale e' sempre collocato all' inizio di un blocco da 1 K

5 -- Se i contenuti dei bits 2 e 3 del registro di

controllo 36869 sono entrambi a 0 (zero) allora l' inizio della memoria viene spostato nella posizione standard $8000

Il generatore di caratteri definibile e' molto importante, non solo perche' consente di creare speciali caratteri grafici ma permette anche l' uso dell' alta risoluzione sul VIC.
La risoluzione consentita e' di 176*184 punti sufficiente per una buona qualita' dell' immagine.

Il primo passo per creare un nuovo gruppo di caratteri e' di predeterminare un blocco di memoria RAM per l' immagazzinamento.
Se i nuovi caratteri che si vogliono definire possono essere rappresentati da una matrice di 8*8 allora saranno richiesti, per tutti, 2048 Bytes.
Se invece si vogliono rappresentare caratteri con risoluzione 8*16 allora saranno necessari 4096 Bytes di memoria.
E' chiaro che in una configurazione standard del VIC che ha solo 3584 Bytes liberi, sara' possibile di definire SOLO la prima ipotesi e cioe' i caratteri con matrice di 8*8.

La memoria utente su un VIC standard inizia dalla locazione 4096 e termina alla locazione decimale 7679
Poiche' come abbiamo detto il generatore di caratteri programmabile puo' iniziare da uno dei seguenti punti di memoria:

4096

5120

6144

7168

e poiche' deve iniziare dalla locazione piu' alta

disponibile per non trovarsi in confusione con il programma Basic o con i dati, sara' necessario, sempre con riferimento alla configurazione base del VIC, far partire il generatore dall' indirizzo decimale 5120
Con questo sistema resteranno ancora liberi 1024 bytes.

E' necessario sottolineare che qualora si voglia operare sul generarore di caratteri programmabile e' indispensabile disporre di una espansione di almeno 3 K in quanto 1K di memoria disponibile e' veramente poco

Restando comunque nella configurazione standard, il secondo passo sara di abbassare i puntatori della memoria per evitare soprascritture sui caratteri. Questo potra' essere ottenuto con le seguenti istruzioni da ripetere tutte le volte che si dia un reset alla macchina sia via Hardware, mediante spegnimento, che Software.
Le istruzioni potranno essere le seguenti:

POKE 51 255 POKE52 19
Sposta l'inizio delle stringhe a 5119

POKE55,255 POKE56,19
Sposta la fine della memoria sempre a 5119

Successivamente sara' necessario inserire i valori di ogni carattere programmato dentro le specifiche locazioni di memoria per mezzo di comandi POKE o tramite linguaggio macchina.

Nella tavola seguente forniamo un esempio di transcodifica con i rispettivi valori, mentre in Appendice riportiamo un breve programma per generare i caratteri, per salvarli su nastro o su disco, per stamparli, ecc.

Dopo che un carattere e' stato disegnato con gli schemi mostrati dovra' essere convertito in un blocco di otto valori numerici ed inserito nella memoria RAM assegnata.
Ogni linea della griglia corrisponde ad un Byte ed ogni punto della griglia corrisponde ai vari bit del Byte considerato.
Come si vede dalle tavole i punti vuoti corrispondono ad uno 0 mentre i valori dei punti pieni della griglia variano in rapporto alla posizione sulla riga.

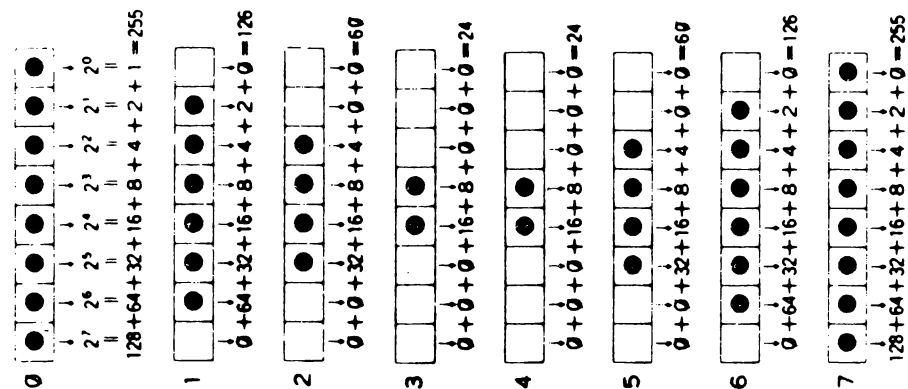
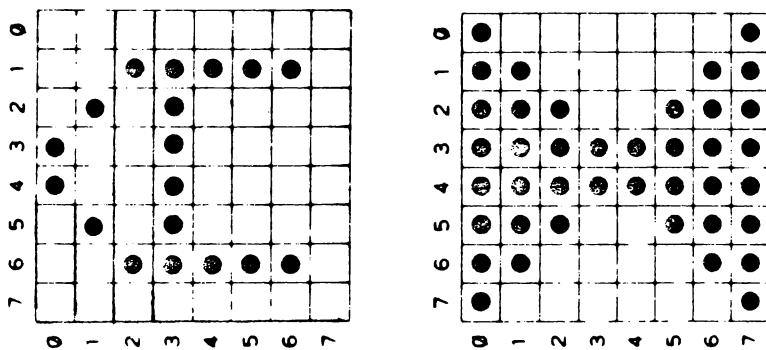
Per una corretta applicazione di quanto detto e' necessario partire dalla prima locazione della memoria riservata.
Per comodita' riportiamo alcuni esempi di programmazione in Basic sempre partendo dal presupposto di operare con un VIC in configurazione base e partendo quindi a caricare i caratteri dalla locazione decimale 5120 dopo aver sistemato i puntatori della memoria.

5120 – 0
5121 – 24
5122 – 60
5123 – 126
5124 – 255
5125 – 24
5126 – 36
5127 – 66

Character # 1

5128 – 24
5129 – 20
5130 – 20
5131 – 18
5132 – 48
etc

Character # 2

UTILIZZC DI CARATTERI SPECIALI PER UN GENERATORE DI CARATTERI

CONVERSIONE DEI CARATTERI IN VALORI NUMERICI

ESEMPIO DI SCHEMATIZZAZIONE DEI CARATTERI

```
100 FOR I = 0 TO 2048
110 READ A$
120 IF A$ = "*" THEN 150
130 POKE 5120 + I, VAL(A$)
140 NEXT
150 END
1000 DATA.......................
1010 DATA .......................
2000 DATA *
```

Nella maggior parte delle applicazioni non e' perco' richiesto di riprogrammare l' intero SET di caratteri, ma si potranno utilizzare i caratteri alfabetici o numerici del generatore standard presente su ROM.

Tutti i caratteri alfanumerici piu' i caratteri grafici del VIC sono contenuti nei primi 128 blocchi ricordiamo ciascuno di 8 Bytes, del generatore di caratteri standard, mentre gli altri 128 blocchi contengono il reverse dei primi.
Per questo ad esempio si possono trasferire su RAM direttamente i 128 caratteri ed adoperare la parte che sarebbe riservata ai REVERSE per inserire dei caratteri speciali.

Per trasferire i 128 caratteri da ROM a RAM si potra' usare la seguente Routine:

```
100 FOR I = 0 TO 1024
110 POKE 5120 + I, PEEK (32768 + I)
120 NEXT I
```

Questa routine lascia come detto la possibilita' di definire gli altri 128 caratteri partendo dall' indirizzo decimale 6155 come descritto nel primo esempio.

Sempre a titolo esemplificativo riportiamo una

piccolo programma che carichi i primi 128 caratteri dal Generatore standard e programmi gli altri.

```
10  POKE 51,255   POKE 52,19
20  POKE 55 255 : POKE 56,19
30  CLR
100 FOR I = 0 TO 1024
110 POKE 5120 + I,PEEK (32768 + I)
130 NEXT I
200 FOR K = 0 TO 1024
210 READ A$
220 IF A$ = "*" THEN 2000
230 POKE 6154 + I, VAL(A$)
240 NEXT
1000 DATA......................
1010 DATA......................
.... DATA......................
1900 DATA *
2000 END
```

Desiderando utilizzare la seconda parte del generatore di caratteri, cioe' quella che parte dala locazione $8800 (decimale 34816) sara' necessario eseguire le seguenti istruzioni per posizionare correttamente il registro di controllo n 2 del 6561

```
4 POKE 36869 253
5 POKE36869 PEEK(36866) OR 128
```

E importante ricordarsi di salvare su nastro o su disco i programmi usati per definire i nuovi caratteri.

Per reinserire il normale funzionamento del generatore di caratteri standard e' necessario usare le due istruzioni seguenti:

```
1 POKE 36869 240
2 POKE 36866 150
```

Se questi nuovi caratteri sono usati con comandi di POKE e quindi incorporati in una stringa e' importante conoscere quale e' il carattere rimpiazzato.
Cio' puo' essere determinato usando la tavola dei codici ASCII del VIC (nel manuale d'uso) per vedere a quale carattere si fa riferimento.
Questo perche quando si scrivono i programmi si scrive nella stringa il vecchio carattere, mentre quando il programma andra' in esecuzione automaticamente sara' cambiato.

E' importante notare anche che quando si usano i comandi di POKE per generare nuovi caratteri si deve determinare anche il colore in cui si vogliono perche' in caso contrario si avra' una resa di bianco su bianco e percio' praticamente invisibile.

D I S P L A Y   F O R M A T   C O N T R O L

La visualizzazione standard sul monitor collegato al VIC e' di 22 righe per 23 colonne con caratteri costituiti da matrici 8*8
Tutti questi valori possono esser variati alterando il contenuto dei relativi registri del 6561.
Ci sono cinque formati di schermo che possono essere usati dall' utente.

1 - Posizione della prima colonna di caratteri alla sinistra dello schermo.

Questa posizione puo' essere cambiata alterando i contenuti del Registro di controllo n 1 posto alla locazione 36864, il cui normale valore espresso in decimale e controllabile tramite un istruzione di PEEK. e' 12.
Incrementando il valore del registro di due la posizione della prima colonna di caratteri si muove verso destra dello spazio di un carattere.
Il valore minimo da porre nel registro di cui alla posizione 36864 e' 0, il massimo dipende dalla larghezza dello schermo e va da 22 con una larghezza di schermo di 22 caratteri a 64 con una larghezza di schermo di una colonna.

2 - Posizione della prima riga di caratteri dal punto piu' alto dello schermo.

Anche questa posizione puo' essere cambiata variando il contenuto del registro di controllo n 2 in posizione di memoria 36865 il cui valore normale e' di 38.
Incrementando il valore nel registro con passo 4 l' area di schermo e' mossa in basso di una linea per volta.
Anche qui il minimo valore attribuibile al registro di 36865 e' 0 mentre il massimo e' di 255.

Occore tener presente tuttavia che un valore maggiore di 130 portera' ad una sparizione dello schermo e potra' essere usato come un blanking.

3 - Determina il numero di righe nello schermo.

Il numero di righe che appaiono e' determinato dal valore immagazzinato nei bits da 1 a 6 del registro di controllo n 4 localizzato all' indirizzo 36867 ed il cui valore standard e' 174.
Il valore eventualmente da immagazzinare in questo registro e' ottenuto moltiplicando il numero di righe desiderato per 2 ed aggiungendo al valore 128
Infatti ripetendo questa operazione a titolo di prova si ottiene che lo standard 174 anzidetto e' dato da:

23 (numero di righe) * 2 =46 + 128 =174

Il numero minimo di righe e' di 1 il che dara' al registro il valore di 130 per la suddetta formula.
Il numero massimo di righe e' 32 che si ottiene ponendo il valore 192 nel registro di controllo n 2 e restringendo in proporzione lo schermo.
Si ricorda comunque che la memoria di schermo NON deve superare i 506 Bytes perche' cio' porterebbe danni alla parte del sistema operativo che serve per gestire lo schermo.

4 - Determina il numero di colonne sullo schermo.

Il numero di colonne che appaiono sullo schermo e' determinato dai bits da 1 a 6 del registro di controllo n 3 alla locazione 36866 il cui contenuto e' normalmente 150.
Il numero immagazzinato in questo registro e' ottenuto il numero di colonne desiderato al valore 128
Anche qui il numero minimo di colonne e' 1 ottenuto ponendo il valore del registro detto a 129.

Il numero massimo e'27 colonne con valore da immettere di 155 tenendo sempre presente che pero' si dovra' accorciare in proporzione il numero delle righe sempre per non superare i 506 Bytes di memoria di schermo.

5 - Determina la misura di ogni matrice che genera il singolo carattere.

Tutti i caratteri sono normalmente visualizzati come una matrice di 8*8 punti.
Cambiando pero' il valore del bit 0 del registro n 4 posto all' indirizzo di memoria 36867, si puo' ottenere una matrice di 8*16.
Il sistema e' abbastanza semplice in quanto si tratta solo di aggiungere 1 al corrente contenuto del registro detto o di sottrarre 1 per riportarlo alla primitiva posizione.

Tutto cio' si dimostra di grande utilita' quando si desideri lavorare in alta risoluzione.

Per chiarezza diamo qui di seguito una tavola con l'indirizzo e la spiegazione abbreviata, dei vari registri

esa	decimale	descrizione
9000	36864	bit da 0 a 6 , centratura orizz.
9001	36865	centratura verticale
9002	36866	bit da 0 a 6 . numero delle colonne bit 7 fa parte dell'ind. per la matrice vid.
9003	36867	bit da 1 a 6 , numero delle righe bit 0 sceglie i caratt. 8*8 o 16*8
9004	36868	
9005	36869	bit da 0 a 3 . start per gener. di caratt. bit da 4 a 7 , indir. per matrice video

bit 3 2 1 0		ind. di start per gen. carat. esa	decimale
0 0 0 0	rom	8000	32768
0 0 0 1		8400	33792
0 0 1 0		8800	34816
0 0 1 1		8c00	35840
1 0 0 0	ram	0000	0000
1 0 0 1		xxxx	xxxx
1 0 1 0		xxxx	xxxx
1 0 1 1		xxxx	xxxx
1 1 0 0		1000	4096
1 1 0 1		1400	5120
1 1 1 0		1800	6144
1 1 1 1		1c00	7168

esa	decimale	descrizione
9006	36870	posizione orizzontale penna ottica
9007	36871	posizione verticale penna ottica
9008	36872	valore della paddle X
9009	36873	valore della paddle Y
900a	36874	frequenza oscillatore 1 (bassa) da 128 a 255
900b	36875	frequenza oscillatore 2 (media) da 128 a 255
900c	36876	frequenza oscillatore 3 (alta) da 128 a 255
900d	36877	frequenza generatore di rumore bianco
900e	36878	bit da 0 a 3 . volume del suono bit da 4 a 7 . colore ausiliario

900f	36879	registro del video e colore del bordo bit da 4 a 7 colore del sottofondo bit da 0 a 2 . colore del bordo bit 3 , selezione modo normale o inverso

D I S P L A Y   C O L C U R   C O N T R O L

Il Vic ha due modi di operare sui colori:
ALTA RISOLUZIONE

MULTICOLORE

Il modo di operare piu' i colori usati e' determinato dal contenuto dei registri di controllo n 15 (indirizzo 36878) e n 16 (indirizzo 36879) che si riferiscono alla memoria video RAM.
Questa parte RAM della memoria video e' lunga 506 bytes ed ha come inizio la locazione $9600 - 38400 in decimale.
Se invece della configurazione standard del VIC si adopera una espansione di piu' di 8 K in totale allora l' inizio della memoria RAM per il colore viene spostata in basso alla locazione esa $9400 - decimale 37888.
Di questi Bytes in questa parte della memoria ci interessano solo i primi 4 BITS:
BITS da 0 a 2 sono usati per selezionare il colore del carattere.

Il BIT 3 determina se il carattere usato e' in alta risoluzione o in multicolore.

A L T A   R I S O L U Z I O N E

Il modo Alta Risoluzione e' selezionato ponendo a zero il bit 3 della RAM di video colore. Questo e' anche il modo normale di operare.
In questo modo di operare esiste una corrispondenza uno a uno tra i bits del generatore di caratteri ed i punti che appaiono sullo schermo.

Cio' consente che tutti i bits a UNO possano

apparire sullo schermo come punti di un colore, mentre i bits a ZERO appaiono come punti di un'altro colore.
Ogni carattere ha due colori, il colore esterno (FOREGROUND COLOUR) ed un colore interno (BACKGROUND COLOUR) in questo caso rispettivamente evidenziato da tutti i bits a UNO o da Tutti i bits a ZERO.

Ciascuno di questi colori e' determinato dai primi 3 bits del VIDEO COLOUR RAM e l' altro dai bits da 4 a 7 del registro di controllo n 16.
Nel modo normale di operare il colore esterno e' immagazzinato nella RAM mentre il colore interno comune a tutti i caratteri e' dato dal registro di controllo 16.

Cio' puo' essere rovesciato in maniera tale che tutti i caratteri abbiano lo stesso colore esterno determinato dal contenuto del registro 16 ed un colore interno settato dal contenuto della memoria RAM.
Che i caratteri sullo schermo abbiano o meno lo stesso colore interno o lostesso colore esterno dipende dal bit 3 del registro di controllo 16.

Infatti se il bit 3 e' a UNO allora potremo avere caratteri di colori diversi con lo stesso interno.
Se il bit 3 e' a ZERO allora tutti i caratteri avranno lo stesso colore ma con un intervno differente.

In aggiunta a tutto cio', selezionando i bit da 0a 2 del registro di controllo n 16 del 6561 sara' possibile cambiare i colori del bordo del monitor.

I colori che possono esser visualizzati con il VIC si possono dividere in due gruppi.
Il primo di otto colori puo' essere usato per il bordo del monitor e per l' esterno dei caratteri.

Il secondo gruppo ha sedici colori che possono essere usati per l' interno del monitor e per i colori ausiliari come vedremo nel modo MULTICOLOUR.

Il primo gruppo e' formato dai seguenti colori che riportiamo anche con il termine inglese:

0	NERO	BLACK
1	BIANCO	WHITE
2	ROSSO	RED
3	CYAN	CYAN
4	MAGENTA	MAGENTA
5	VERDE	GREEN
6	BLEU	BLUE
7	GIALLO	YELLOW

Il secondo gruppo o colori ausiliari o interni e' come segue:

0	NERO	BLACK
1	BIANCO	WHITE
2	ROSSO	RED
3	CYAN	CYAN
4	MAGENTA	MAGENTA
5	VERDE	GREEN
6	BLEU	BLUE
7	GIALLO	YELLOW
8	ARANCIO	ORANGE
9	PAGLIERINO	LIGHT ORANGE
10	ROSA	PINK
11	VIOLA	LIGHT CYAN
12	VIOLETTO	LIGHT MAGENTA
13	VERDINO	LIGHT GREEN
14	AZZURRO	LIGHT BLUE
15	GIALLINO	LIGHT YELLOW

In sintesi il MODO Alta Risoluzione usabile e' come segue:

1 - Manipolazione del bit 3 del registro 16 per avere un interno o un esterno comune per il carattere:

per esterno comune - POKE36879,PEEK(36879) AND 247

per interno comune - POKE36879,PEEK(36879) OR 8

2 Manipolazione dei colori attraverso l' uso dei bits 4-7 del registro 16.
Ci sono sedici possibili colori, c'e' il numero di colore come spiegato nella precedente tavola che puo' essere settato con i comandi spiegati nell' esempio sottoriportato:

```
POKE 36879,PEEK(36879) AND 15
POKE 36879,PEEK(36879) OR (C*16)
```

Dove C e' il colore scelto della tavola.

Per ripristinare lo standard:
POKE36879,27

3 - Manipolazione dei colori relativamente al bordo dello schermo.

Come detto bisogna operare sui Bits 0-2 del registro di controllo n 16.
Ci sono otto possibili colori del bordo come spiegato prima e con riferimento alla prima tabella ed utilizzando sempre la variabile C come scelta del colore si opera come segue:

```
POKE 36879,PEEK(36879) AND 248
POKE 36879,PEEK(36879) OR C
```

M O D O   M U L T I C O L O R E

Il mod multicolore e' selezionato mettendo a 1 il bit 3 del VIDEO COLOUR RAM.
In questo modo esiste una corrispondenza due a uno tra Bits del generatore di caratteri e punti visualizzati sullo schermo.
Questo consente che due Bits della matrice del generatore di caratteri per quel codice di carattere corrisponda ad un punto sullo schermo e che il colore di quel punto sia determinato per mezzo di un codice a due bits del generatore di caratteri.
Cosi' mentre nel modo Alta Risoluzione si possono adoperare due colori per ogni carattere, nel Multicolor questi colori diventano quattro.
Tuttavia poiche' due bits del generatore di caratteri corrispondono ad un singolo punto sullo schermo la risoluzione orizzontale e' la META' di quella che si !ttiene nel MODO ALTA RISOLUZIONE
Ogni carattere occupa lo stesso spazio in tutti e due i modi, per cui entrambi i modi possono essere usati e per di piu' interconnessi con l' avvertenza che un punto in MULTICOLORE occupa 2 spazi rispetto allo stesso punto in ALTA RISOLUZIONE.
Il modo MULTICOLORE non e' consigliabile per l'uso con il generatore di caratteri standard su ROM, ma puo' dare grandi soddisfazioni impiegandolo con un generatore di caratteri programmabile su RAM.
Questo perche il generatore standard e' disegnato per l'uso del MODO IN ALTA RISOLUZIONE dove ogni bit rappresenta un punto della matrice relativa a quel carattere.
L' uso del modo multicolore puo' essere sintetizzato come segue.
1 - Selezionare il colore interno su uno dei sedici clori della seconda tabella operando come segue:

```
POKE 36879, PEEK (36879) AND 15
POKE 36879  PEEK (36879) OR (C*16)
```

Sempre con C come variabile di colore.

2 - Selezionare il bordo esterno in uno degli otto colori consentiti ed immagazzinando il valore sempre nella variabile C:

```
POKE 36879, PEEK (36879) AND 248
POKE 36879  PEEK (36879) OR C
```

3 - Selezionare uno dei sedici colori ausiliari sempre con C come variabile di colore come nell' esempio:

```
POKE 36879, PEEK (36879) AND 15
POKE 36879, PEEK (36879) OR (C*16)
```

La tabella che segue puo' essere utilizzata in un generatore di caratteri su RAM con inizio a 5120. Il carattere ha un codice di valore 0 e mostra l' impiego di ognuno dei quattro colori ammessi nel MULTICOLOUR MODE.

BYTE	Bit								esa	Locazione
	7	6	5	4	3	2	1	0		
0	0	0	0	1	1	0	1	1	1b	5120
1	0	0	0	1	1	0	1	1	1b	5121
2	0	0	0	1	1	0	1	1	1b	5122
3	0	0	0	1	1	0	1	1	1b	5123
4	0	0	0	0	0	0	0	0	00	5124
5	0	1	0	1	0	1	0	1	55	5125
6	1	0	1	0	1	0	1	0	aa	5126
7	1	1	1	1	1	1	1	1	ff	5127

Gli effetti sonori del VIC sono controllati tramite 5 registri di controllo del 6561.
Quattro di questi registri sono associati con il generatore sonoro, mentre il quinto controlla il volume dell' uscita del suono.
Ognuno dei quattro registri del generatore sonoro e' collegato ad un oscillatore ed i contenuti dei registri determinano la frequenza di uscita dell' oscillatore. La frequenza e' determinata variando la larghezza della vibrazione. L' uscita da tutti e quattro gli oscillatori e' un' onda simmetrica quadra.
Le uscite sono combinate per dare un segnale di ingresso audio al monitor, dove il suono e' generato tramite il normale circuito di speaker.
Uno dei quattro oscillatori funziona per il rumore di fondo mentre gli altri quattro generano un tono semplice.

REGISTRI DI CONTROLLO PER IL SUONO

L'audio oscillatore n 1 e' controllato dal registro di controllo n 11 di locazione 36874.Il bit 7 mette in ON questo oscillatore se e' a UNO, mentre lo mette i n OFF se e' a 0.I bits da 0 a 6 controllano la frequenza.
Base per una bassa frequenza.
Immettendo il valore di 128 in questo registro sara' emesso il suono di piu' bassa frequenza realativamente agli oscillatori.Si avranno cioe' dei suoni in bassa frequenza.E' normalmente denominato in termini musicali come generatore BASE.

Audio oscillatore n 2
E' controllato dal registro n 12 di locazione 36874.
Il bit 7 ha la stessa funzione che nel precedente registro.

I bits da 0 a 6 controllano la frequenza.
Si avranno con questo registro i suoni di frequenza intermedia fra i due oscillatori 1 e 3.
E normalmente denominato in termini musicali come generatore TENORE.

Audio oscillatore n 3
Controllato dal registro di controllo n 13 di locazione decimale 36876.
Il bit 7 lo attiva se e' a UNO o lodisattiva se e' posto a 0.
I bits da 0 a 6 ne controllano la frequenza.
Questo registro controlla la possibilita' di suoni in alta frequenza.
Per questo oscillatore sempre in termini musicali si puo' adottere la definizione di SOPRANO.

Generatore del rumore di fondo.
Controllato dal registro n 14 di locazione 36877.
Funzionamento del bit 7 come sopra.
I bits 0-6 controllano la frequenza base di questo generatore.
Questo e' un generatore di rumore di fondo BIANCO che da una sequenza di vibrazioni prodotte casualmente ed associate ad una frequenza determinata dal contenuto del registro di controllo.

Controllo volume
Registro di controllo n 15 di locazione 36878.
Quando uno o piu' oscillatori sono attivati manipolando questo registro e' possibile determinare il volume tramite i bits 0-3

I valori da immettere nei registri degli oscillatori sono riportati nella tavolaseguente.
Dato che non e' il luogo per una approfondita discussione sulla musica del VIC perche' porterebbe via spazio eccessivo abbiamo ritenuto piu' opportuno allegare in appendice una serie di piccoli programmi musicali a titolo di esempio.

TAVOLA DEI VALORI E DELLLE NOTE

I valori di questa tavola di cui si leggono accanto le note possono essere immessi nei registri di controllo relativi agli oscillatori tramite un comando di POKE.
Come si puo' vedere dai programmi possono essere adoperati contemporaneamente piu' oscillatori .

NOTA MUSICALE	VALORE
C	128
C£	134
D	141
D£	147
E	153
F	159
F£	164
G	170
G£	174
A	179
A£	183
B	187
C	191
C£	195
D	198
D£	201
E	204
F	207
F£	210
G	213
G£	215
A	217
A£	219
B	221
C	223
C£	225
D	227

D£	228
E	230
F	231
F£	232
G	234
G£	235
A	236
A£	237
B	238
C	239
C£	240

LA CASSETTA DI REGISTRAZIONE

Il Vic puo' avere come unita' esterna una cassetta di registrazione per immagazzinare dati e/o programmi.

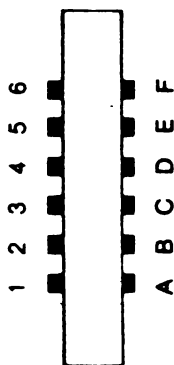
L' unita' a cassette e' connessa al Vic tramite un connettore a 6 vie:

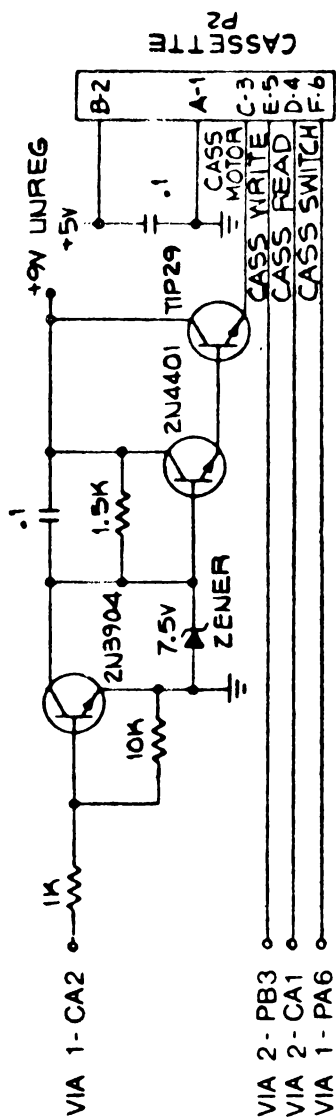
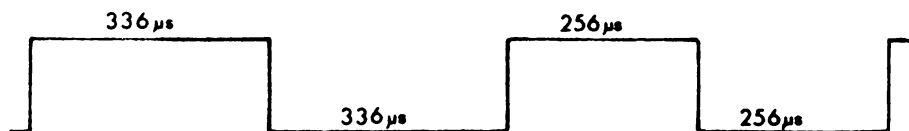
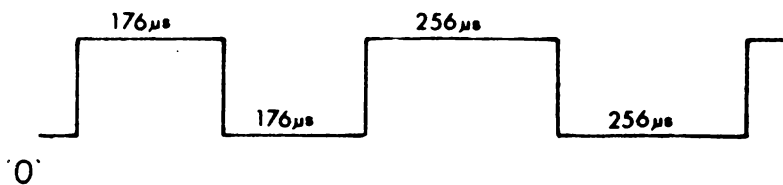
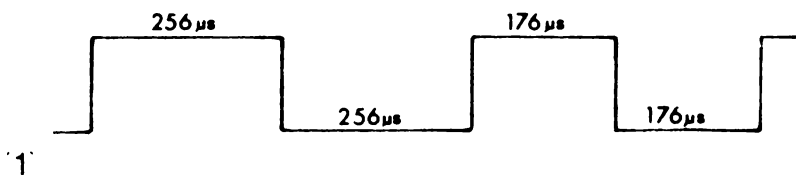
A-1	GND
B-2	+ 5 VOLTS
C-3	MOTORE
D-4	LETTURA
E-5	SCRITTURA
F-6	SWITCH SENSE

L'alimentazione del motore e' connessa tramite tre transistors, i quali con circa 500 mA di uscita permettono al 6522 (VIA 1) di pilotarlo direttamente.
Il motore puo' essere attivato o disattivato operando sulla linea CA2 del 6522 n 1 come segue:

POKE37148,PEEK(37148) AND 241 OR 14 che attiva il motore

POKE37148,PEEK(37148) OR 12 AND NOT 12 che lo disattiva.

Il verso della linea di input, linea PA6 sul VIA n 1, e' connesso ad uno SWITCH sulla piastra della cassetta e che controlla quando uno dei pulsanti, PLAY,REWIND,FAST FORWARD e' premuto.

PIN #	TYPE
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SWITCH

PIEDINATURA DEL CONNETTORE DELLA CASSETTA

CIRCUITO DI INTERFACCIA TRA IL 6522 E LA CASSETTA

SEGNALI D'USCITA PER IL REGISTRATORE

O P E R A Z I O N I   S U L L A   C A S S E T T A .

Nel normale uso alla cassetta e' assegnato come device il n 1.
Si ricorda che il numero di device assegnato oppure aperto in quel momento e' individuabile nella locazione di memoria 186 della pagina zero.
Il numero assegnato al device, il numero di file logico e l' indirizzo secondario sono usati per salvare o ritrovare files di dati nell' unita'.

Il numero di file logico assegnato puo' essere un qualsiasi numero fra 1 e 255 ed il suo uso consente di manipolare piu' files contemporaneamente sullo stesso device.
Con cio' si comprende come la sua importanza di impiego sia piu' per il floppy disk che per l' unita' a cassette.
Il numero di file logico sul quale in quel momento si sta operando e' immagazzinato nellla locazione 184 sempre della pagina zero.

L' indirizzo secondario e' importante perche' determina il modo di operare su quel determinato file.
Il valore dell' indirizzo secondario e' immagazzinato nella locazione 185 per quanto riguarda l' unita' a cassette e di norma e' settato a zero.

Se l'indirizzo secondario e' zero allora vuol dire che il nastro e' aperto per una operazione di lettura.
Se e' settato a UNO allora e' consentito una operazione di scrittura.
Se invece il valore presente nell' indirizzo 185 e'settato a due allora il nastro sara' aperto in scrittura ed un blocco di fine nastro sara' inserito al momento della chiusura del file.

Il sistema operativo del VIC e' configurato per consentire che due diversi tipi di files possano essere immagazzinati su cassetta:

FILES DI PROGRAMMI

FILES DI DATI

Questo discorso e' tuttavia abbastanza una convenzione poiche' un programma puo' essere archiviato come file di dati e viceversa.
La differenza nei due tipi di files non e' nella loro applicazione ma nel sistema in cui il contenuto della memoria viene immagazzinato sul nastro.

A questo punto sara' importante parlare dei files di tipo BINARIO e dei files di tipo ASCII.

I FILES BINARI

Il file di tipo binario e' normalmente usato per immagazzinare programmi.
Questo tipo di file e' creato dal sistema operativo per immagazinare il contenuto della memoria dell' unita' centrale da una certa locazione di partenza ad un' altra locazione di fine.
E' chiamato file binario perche' immagazzina su nastro il valori binario con il quale quelle specifiche locazioni di meoria sono immagazzinate.
Come abbiamo detto i comandi del Basic sono immagazzinati in memoria come dati di un Byte.
L' uso di questo particolare sistema porta al fatto che questi comandi non siano presenti nella memoria come appaiono sullo schermo o come vengono digitati da tastiera, ma vengono appunto ridotti ad un solo Byte.

Essendo quindi codificato, questo sistema e' il piu' efficente dal punto di vista della ottimizzazione dello spazio.

E' essenziale notare che i programmi o le routines in linguaggio macchina non possono essere salvati su nastro altro che con questo sistema.

L' indirizo di partenza dal quale un file binario sara' registrato e' contenuto nelle locazioni di memoria 172 e 173.
Quando si inizia il processo di registrazione o SAVE entra in funzione la routine di SAVE del sistema operativo che andra' a controllare i valori delle locazioni di memoria citate per vedere da che punto deve incominciare a registrare .

Queste locazioni avranno di norma caricati i valori 0 e 4 per comunicare al sitema che l' inizio del testo da registrare incomincia in 1024.
Per registrare valori o dati che si vuole inizino da un' altro punto della memoria invece che da 1024 bastera' semplicemente cambiare i valori dei due puntatori di inizio posti a 172 e 173.

I puntatori di fine testo sono localizzati in 174 e 175 e sono settati per riconoscere un indirizzo contenente un doppio zero che comunica alla macchina la fine di un programma o meglio la fine dell' indirizzo di link.

E' da notare pero' che volendo variare sia l' indirizzo di partenza sia non far riconoscere quello di fine sara' necessario scrivere una piccola routine in linguaggio macchina o adoperare il MACHINE LANGUAGE MONITOR perche' la routine di SAVE resetta i puntatori allo standard tutte le volte che viene chiamata in azione.

I FILES ASCII

I files ASCII sono normalmente usati per immagazzinare dati anche se possono essere usati per immagazzinare programmi come si vedra' con la procedura di MERGE.
Il contenuto di questi Files e' lostesso di quello che appare sullo schermo o che viene digitato da tastiera. Non esiste cioe' una codifica preventiva come per i comandi del Basic.

I files ASCII pero' possono essere creati e letti solo attraverso comandi inseriti in un programma Basic.
I files binari invece possono essere letti o scritti tramite comandi diretti di LOAD e SAVE anche se e' bene ricordare che questi comandi possono essere inseriti in un programma Basic.

Un file Ascii deve essere aperto con un comando di OPEN che dovra' specificare:

IL FILE LOGICO

IL NUMERO DI DEVICE

L INDIRIZZO SECONDARIO

IL NOME DEL FILE

Il sistema operativo del VIC interpreta questi parametri e consente all' utente di leggere o scrivere il file sulla periferica desiderata.

I files binari sono caricati come successive locazioni di memoria mentre un file ASCII e' caricato come una stringa di variabili.

Per immagazzinare in questo modo le stringe si dovrebbe stare ad accendere e spengere continuamente l' unita' a nastro.
Il VIC persuperare questo ha creato un buffer di cassetta di 192 Bytes in cui i dati sono scritti o letti prima di passare dalla memoria centrale al nastro o viceversa.
Per cui solo quando il buffer e' temporaneamente pieno allora il motore si ferma da se'.
I dati sono immagazzinati su nastro in blocchi di 192 Bytes ed il motorino, fra un blocco e il successivo, si ferma per 2 secondi non per ricaricare i dati sul buffer in quanto questo avviene in tempi di millisecondi ma per accellerare e decelerare la corsa del nastro.

L' indirizzo di partenza del buffer di cassetta e' alla locazione 828 mentre il puntatore di partenza e' localizzato negli indirizzi 178 e 179.
Il numero di caratteri immagazzinato nel buffer e' dato dal valore della locazione 166.
Questa locazione puo' essere utilizzata dal programmatore per controllare l' ammontare di spazio lasciato da un file di dati.

Sia che si immagazzini un file di tipo binario che un file ascii si fa sempre ricorso al particolare metodo di memorizzazione e di ricerca messo a punto dalla Commodore che consente il massimo della sicurezza sia in fase di lettura che di scritture.
Ogni Buyte di dati o di programmi e' codificato dal sistema operativo usando 3 distinte audio frequenze con i seguenti valori:

SEGNALE LUNGC CON FREQUENZA 1488 Hz

SEGNALE MEDIO CON FREQUENZA 1953 Hz

SEGNALE CORTO CON FREQUENZA 2840 Hz

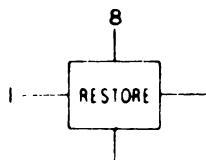
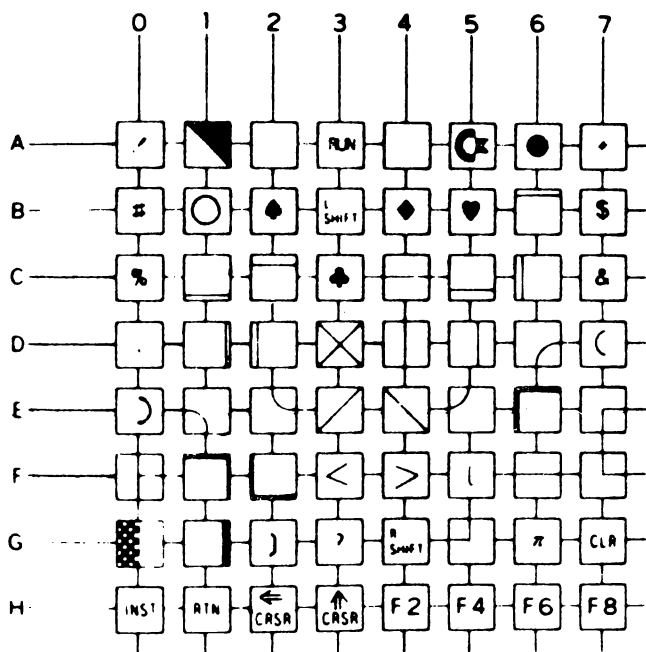
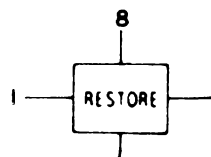
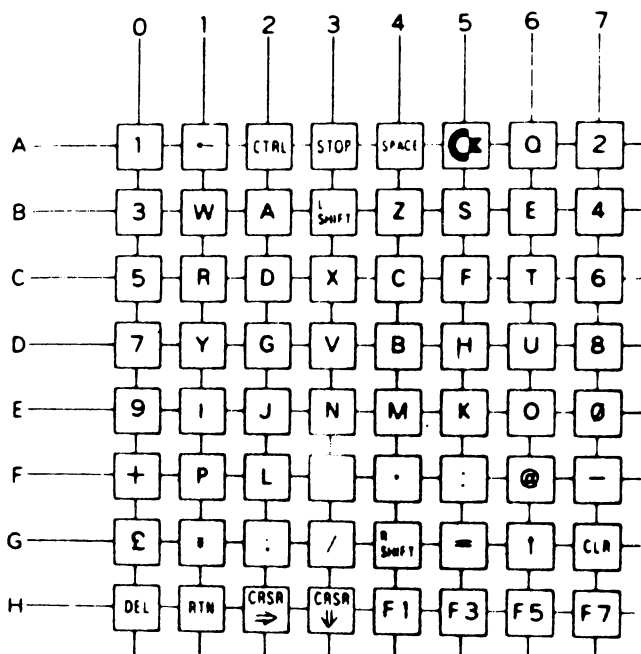
Queste frequenze sonc onde quadre con un ciclo di 256 microsecondi a livello alto (1) e 256 microsecondi a livello basso (0)

Poiche' un byte di dati e' memorizzato in appena 8.96 millisecondi, l' intero buffer di dati in un file ASCII dovrebbe essere memorizzato in appena 1.7 secondi.
Ad un controllo accurato si vede tuttavia che il tempo necessario e' di 5.7 secondi.
Questa discrepanza nel tempo e' dovuta a due fattori.
La prima e' che per ridurre la possibilita' di errore ogni dato viene memorizzato due volte e la seconda e' che fra ogni blocco f di dati viene inserito un INTER-RECORD GAP di 2 secondi.

L' uso apparentemente sovrabbondante di tecniche di controllo di errore in realta' si dimostra utilissimo in fase operativa.
Ci sono due livelli di controllo di errore.
Il primo divide i dati in blocchi di otto bytes e poi calcola un nono bytes chiamato CHECKSUM DIGIT.

L A T A S T I E R A

La tastiera del Vic ha un totale di 66 tasti (KEYS).
Questi comprendono 64 tasti alfanumerici e tasti speciali di funzione, il tasto RESTORE ed il tasto di SHIFT LOCK.
I 64 tasti sono connessi a otto linee di I/O dell' integrato VIA n2.
Il tasto di RESTORE e' collegato alla CA1 del VIA n 1 ed e' usato per generare un NMI (NON MASKABLE INTERRUPT) del processore.
Il tasto di SHIFT LOCK ha semplicemete funzioni meccaniche di riconoscere che il tasto SHIFT appunto e' sempre in pressione.
La scansione della matrice della tastiera avviene per mezzo della pressione di un tasto il quale connette una delle otto linee di input con una delle otto linee di output controllate dal VIA(1) 6522.
La scansione della matrice della tastiera e il controllo di tasto premuto e' posto sotto controllo software dal sistema operativo.
L' intera temporizzazione del processo non puo' essere dedicata alla scansione della tastiera fino a quando questa non sia inizializzata da un interrupt di 1/60 di secondo.
La scansione della tastiera e' una delle funzioni della IRQ (interrupt servicing routine) mentre l' interrupt di 1/60 di secondo e' fornito dal timer N. 1 del VIA(2).
L' IRQ inizia alla locazione $EABF e la scansione della tastiera a $EB1E.
La routine di scansione della tastiera passa attraverso una sequenza di operazioni il cui risultato, cioe' ogni carattere in INPUT, viene inserito nel KEYBOARD BUFFER. La sequenza e' la seguente :

TASTIERA DEL VIC (SHIFTATA E NON)

connessioni della tastiera al 6522

1) Controllo di tasto premuto, se negativo uscire dalla Routine

2) Inizializza le porte di I/O del VIA(2), per la scansione della tastiera e fissa i puntatori nella tavola N. 1 della tastiera, fissa il contatore di caratteri a 0

3) Fissa una linea della porta B bassa e controlla l' input del carattere sulla porta A esegue otto SHIFT a destra del contenuto del registro della porta A, se il CARRY e' settato il carattere e' presente. Ogni SHIFT incrementa il contatore di caratteri e lo immagazzina nel registro Y

4) Ritorna al passo 3 e ripete la operazione per la successiva colonna, se il carattere e' trovato allora continua.

5) Usa il valore del contatore di caratteri come puntatore indice alla tavola dei caratteri della tastiera per generare il codice ASCII corrispondente al tasto premuto.

6) Controlla se e' stato premuto il tasto SHIFT o STOP

7) Valutazione della funzione di SHIFT:

Se il tasto shift e' premuto usa il contatore di caratteri per accedere alla tavola N. 2 della tastiera

- Se e' premuto il tasto CBM usa il contatore dei caratteri per accedere alla tavola N. 3 della tastiera

- Se sono stati premuti i tasti SHIFT e CBM usa il contatore dei caratteri per accedere alla tavola N. 4 della tastiera.

8) Usa il valore del conatore dei caratteri come puntatore indice alla tavola di caratteri della

tastiera descritta al passo 7

9) Controllo per l' operazione di repeat

10) Controllo per i tasti editor dello schermo ed esecuzione dell' azione relativa

11) Esegue il repeat se richiesto

12) Inserisce il carattere ASCII ottenuto dalle tavole dei caratteri della tastiera nel KEYBOARD BUFFER e incrementa il puntatore entro lo stesso Buffer

I contenuti dei dieci caratteri immagazzinati nel buffer della tastiera sono resi disponibili con il metodo FIFO (first in-first out) dalle routines di gestione dello schermo.
Queste routines prendono il primo carattere del buffer della tastiera decrementando il suo puntatore e spostando a sinistra di uno i caratteri del buffer, lasciando spazio per l' inserimento di un nuovo carattere. L' esatta funzione delle routines di gestione dello schermo dipende dal modo di operare del VIC.
Se il VIC e' in modo diretto allora l' input da tastiera fa parte della routine basic che riceve una linea di programma dalla tastiera l' indirizzo di start di questa routine e' alla locazione $C560.
Se sul VIC sta girando un programma BASIC l' input da tastiera fa parte della routine che inizia alla locazione $CBBF ESEGUE INPUT.
Nel MODO DIRETTO i caratteri sono prelevati dal buffer di tastiera e visualizzati sullo schermo, il carattere e' anche posto nel Buffer di 88 bytes del basic.
Queste operazioni vengono ripetute fino a quando non e' premuto il tasto RETURN.
L' interprete basic allora controlla il contenuto del BASIC INPUT BUFFER per vedere se il contenuto

e un comando corretto. Se e' corretto esegue il comando se no riporta un messaggio di ? SYNTAX ERROR

Nel MODO PROGRAMMA il carattere e' prelevato dalla coda del buffer quando cio' e' richiesto da un comando INPUT o GET durante l' esecuzione di un programma.

Cio' e' molto utile in quanto permette di velocizzare l' input da tastiera.

Uno dei svantaggi nell' uso dell' INTERRUPT per inizializzare la scansioene di tastiera e' che le routines di INTERRUPT vengono cambiate durante le operazione di I/O tra il processore e la cassetta o le periferiche seriali di I/O.
L' inconveniente e' che l' utente non puo' controllare il sistema se la tastiera e' disabilitata.
Il problema e' stato aggirato controllando solo il tasto STOP connesso direttamente al PB3 del VIA2.
Le routines di I/O eseguono un semplice controllo dell' attivazione del tasto STOP leggendo il registro A del VIA2. Le routines di controllo del tasto di STOP sono separate dalle altre routines di scansione della tastiera e sono divise in due sezioni:

$F755 Questa e' una parte della routtine di temporizzazione che e' chiamata in tutte le routines di IRQ . Essa aggiorna la variabile TI e controlla il tasto di STOP. E per questo che quando si va in lettura da nastro il valore di TI viene alterato erroneamente. Vengono letti i contenuti della porta B del VIA2 per essere sicuri della loro validita' e inseriti nel flag della STOP key a $91

$F770 Questa e' la parte piu' importante della routine del tasto STOP ed e' richiamata da un

indirizzo indiretto immagazzinato in RAM nel vettore che si trova a $0328
Questa routine prende i contenuti della locazione $91 e li confronta con il valore di $FE se e' uguale vuol dire che il tasto STOP e' stato premuto.
Cio' inizializza la pulitura della coda di tastiera e dei canali di I/O prima di restituire il controllo al modo diretto.

$FEA9 Routine di manipolazione dell' interrupt NMI. Cio' e' generato se il tasto RESTORE e' stato premuto infatti la funzione di restore viene eseguita solamente se anche il tasto STOP viene premuto. Se entrambi i tasti sono premuti allora l' interprete basic salta alla locazione $0002

LOCAZIONI DI MEMORIA USATE DALLA TASTIERA DEL VIC.

$91 FLAG DEL TASTO STOP
$C5 INDICE DI SCANSIONE DI UN TASTO
$C6 INDICE DELLA CODA DI TASTIERA
$F5 $F6 INDIRIZZO INDIRETTO DI SALTO AI TASTI SULLE TAVOLE
$0200 $0258 BASIC INPUT BUFFER
$0277-$0280 BUFFER DI TASTIERA
$028A REPEAT (128 tutti i tasti , 0 solo il cursore)
$028D FLAG DEI TASTI SHIFT,CTRL,CBM
$028F $0290 INDIRIZZO INDIRETTO DI SALTO PER LA TAVOLA DELLA TASTIERA

USCITA SERIALE RS-232

Il VIC e' in grado di comunicare con unita' periferiche siano esse stampanti, modem ed altro usando una porta di comunicazioni seriali conosciuta con il nome di RS232 Input/Output port.
Il nome RS232 si riferisce semplicemente ad un prodotto industriale standard messo a punto per le comunicazioni seriali con periferiche di computer.
Una porta seriale di Input/Output consiste di un minimo di 3 linee:

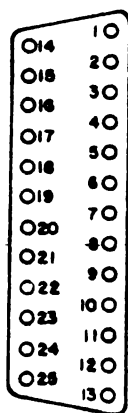
Un output or transmit line.

Una linea di input o di ricezione

Ed una comune linea di massa.

I dati sono trasmessi o ricevuti come un treno di impulsi.
Ad esempio un singolo Byte diventa una stringa o un insieme di 8 impulsi.
Sebbene una porta seriale possa avere appena 3 linee o meglio possa FUNZIONARE con appena 3 linee di solito altre linee sono usate per trasferire e controllare le informazioni.
Il VIC e' capace di ricevere segnali di controllo per consentire una piena " X LINE INTERFACE" altrettanto bene che una semplice "TRHEE LINE INTERFACE".
Quando viene usata la RS232, e vedremo poi il perche' di questo discorso, tutte le linee sono connesse alla porta B del VIA n 1, le stesse linee usate per la user port.
Normalmente una interfaccia RS232 sara' usata per connettere fra loro una porta parallela ed un connetore standard TS232.
L' interfaccia provvedera' anche al Buffer ed all' HIGHER DRIVE VOLTAGE.

PIN	TYPE	NOTE	PIN	TYPE	RS232 FUNCTION
1	GND		A	GND	
2	+5V	100mA MAX	B	CB1	
3	RESET		C	PB0	Sin
4	JOY0		D	PB1	— RTS
5	JOY1		E	PB2	— DTR
6	JOY2		F	PB3	— RI
7	LIGHT PEN		H	PB4	— DCD
8	CASSETTE SWITCH		J	PB5	
9	SERIAL ATN IN		K	PB6	— CTS
10	+9V	100mA MAX	L	PB7	-- DSR
11	GND		M	CB2	— Sout
12	GND		N	GND	— GND

PARTICOLARE DEL CONNETTORE U.PORT PER RS232

PIN		
1	Protective Ground	AA
2	Transmitted Data	BA
3	Received Data	BB
4	Request To Send	CA
5	Clear To Send	CB
6	Data Set Ready	CC
7	Signal Ground	AB
8	Carrier Detect	CF
9	(not used)	
10	"	
11	"	
12	"	
13	"	
14	"	
15	"	
16	"	
17	"	
18	"	
19	"	
20	Data Terminal Ready	CD
21	(not used)	
22	"	
23	"	
24	"	
25	"	

PIEDINATURA SU CONNETTORE STANDARD RS 232

La linea RS232 normalmente trasmette i dati usando un segnale a 12 Volts ,l' implementazione della porta RS232 sul VIC e' molto interessante perche' consente tramite l' uso del Software di emulare una periferica Hardware.
L' hardware in oggetto e' il 6551 UNIVERSAL ASYNCHRONOUS TRANSMITTER AND RECEIVER o abbreviato UART.
I progettisti del VIC intendevano usare questo integrato per generare la porta di ingresso/uscita RS-232, tuttavia la MOS Technology non fu capace di mettere a punto e di consegnare un UART in tempo per l' uscita del VIC.
Si dovette ricorrere pertanto a simulare una RS-232 tramite Software.
Come gli altri integrati di I/O cosi' le funzioni del 6551 sono controllate da registri in locazioni di memoria specifiche.
Gli pseudo registri del 6551 sono localizzati in varie parti dell' area di immagazzinamento variabili nella parte piu' alta della memoria VIC.
Le routines operative cioe' le routines che consentono all' RS-232 di funzionare richiedono 2 buffer ciascuno di 256 Bytes di lunghezza. Uno per ricevere dati , l' altro per trasmetterli.
Questi 512 Bytes di memoria occupati da questi Buffers sono localizzati al punto massimo della memoria RAM disponibile l' indirizzo di partenza di questi 2 buffers e' immagazzinato in 4 registri.
I due piu' importanti registri sono il controllo ed i registri di comando che determinano l' esatta operativita' o le esatte operazioni della porta RS-232.

LO PSEUDO REGISTRO DI CONTROLLO 6551

Locazione esadecimale $0293 - decimale 659
La funzione del registro di controllo e' di fissare la velocita' di trasmissione e di ricezione dei dati e di fissare il numero di Bits necessari per trasmettere ogni carattere.

La velocita' con la quale i dati sono spediti o ricevuti e' chiamata BAUD RATE ed il valore convenzionale per questo e' il numero di Bits per secondo.
Se il BAUD RATE e' fissato a 300 baud ed ogni carattere e' trasmesso come otto bits piu' il bit di STOP ed il bit di Parita' vuol dire che ogni secondo si trasmetteranno 30 caratteri.
La fissazione del BAUD RATE dipende dalle specifiche della periferica che comunica con il VIC tramite l' RS-232, per cui sara' necessario controllare queste specifiche nel manuale della periferica che si desidera collegare.

I bits 5,6,7 di questo pseudo registro controllano il numero di bits necessari per trasmettre o ricevere dati tra il VIC e le periferiche.

LO PSEUDO REGISTRO DI COMANDO DEL 6551

Locazione di memoria esadecimale $0294 - decimale 660

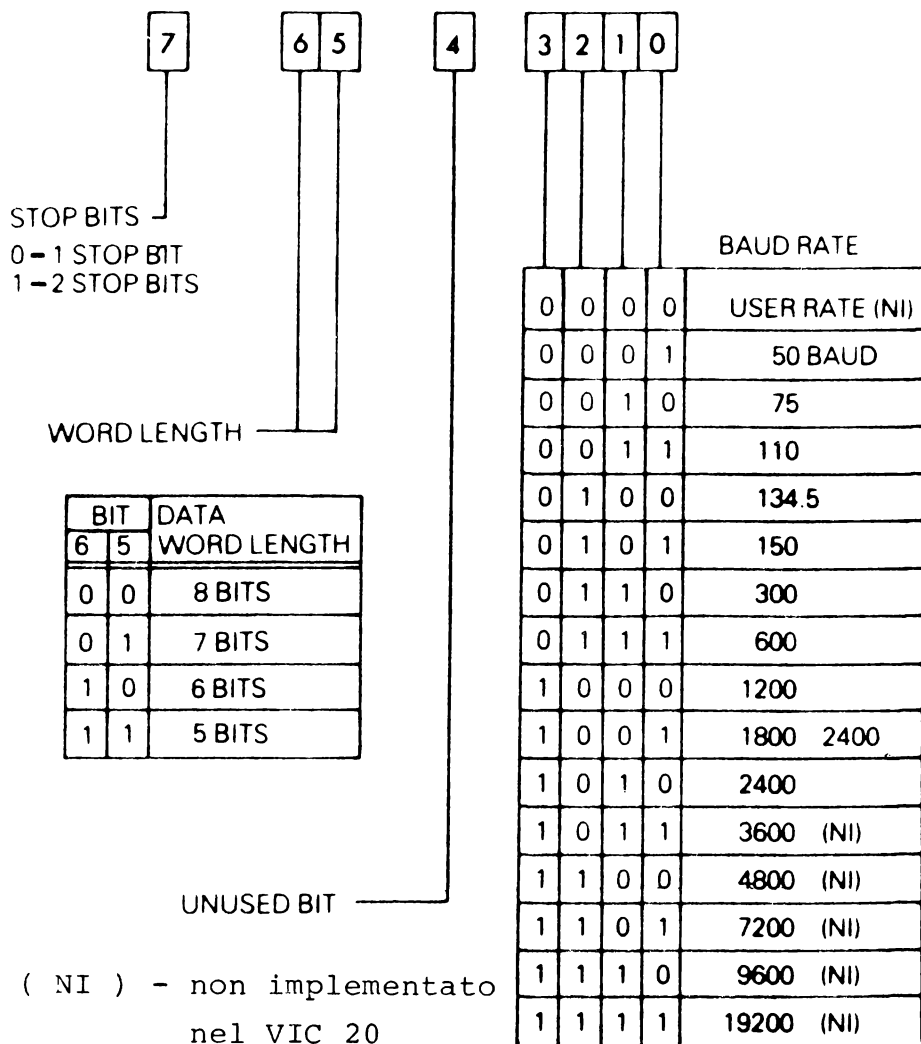
Questo registro di comando controlla il modi di trasmissione e ricezione dati.
Il bit 0 fissa il modo, cioe' un modo a 3 o X linee.
Il bit 4 fissa il cosidetto modo DUPLEX come segue:

DUPLEX PIENO- con trasmissione e ricezione simultanea di dati.

MEZZO DUPLEX- con trasmissione alternata alla ricezione.

I bits 5,6,7 determinano la natura del BIT di parita' e quando sono trasmessi i punti e gli spazi.

BIT 6	BIT 5	DATA WORD LENGTH
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

3	2	1	0	BAUD RATE
0	0	0	0	USER RATE (NI)
0	0	0	1	50 BAUD
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	1800 2400
1	0	1	0	2400
1	0	1	1	3600 (NI)
1	1	0	0	4800 (NI)
1	1	0	1	7200 (NI)
1	1	1	0	9600 (NI)
1	1	1	1	19200 (NI)

REGISTRO DI CONTROLLO RS-232

USO DELLA PORTA RS-232

Apertura di un canale sulla RS232

Punto di entrata per il Codice Macchina esadecimale $FFC0

Comando: OPEN lf,2,0,"rc,rcm"

lf = e' il normale Logical File

rc= per registro di controllo

rcm= per registro di comando

Solo un canale per volta della RS-232 dovrebbe essere aperto.
Poiche' un comando OPEN resetta i puntatori del Buffer, dando un secondo comando di OPEN si distruggera' i dati fissati con il primo comando.
Inoltre questo comando deve essere usato prima di ogni variabile o comando di dimensionamento.
Cio' perche' il comando di apertura del canale della RS232 esegue un CLEAR prima di collocare i 512 Bytes usati dai due Buffers al massimo della memoria.
Nel caso non ci sia sufficiente spazio per i due buffers che ricordiamo sono di 256 Bytes ciascuno si avra' una perdita di dati.
Nel comando OPEN del canale della RS-232 si possono avere, relativamente al nome, fino ad un massimo di 4 caratteri di cui 2 usati dal sistema e gli altri 2 previsti per future espansioni.

Ricevimento dati da un canale RS-232

Comando: GET £ lf,(variabile)

lf = file logico usato dal canale di comando della RS232

$FFC6 - Apre un canale per l' input.

$FFE4 - GET di un carattere dal buffer.

L' input dei dati e' sotto il controllo dei Timers e degli Interrupts del 6522.

LOCAZIONI DI MEMORIA PER LO PSEUDO 6551

$A7	LOCAZIONE TEMPORANEA INGRESSO DATI
$A8	CONTATORE DEI BITS RICEVUTI
$A9	FLAG DEL BIT DI START
$AA	LOCAZIONE DI RICEZIONE DEL BYTE
$AB	IMMAGAZZINAMENTO BIT DI PARITA'
$B4	CONTATORE DEI BIT USCITA
$B5	PROSSIMO BIT DA TRASMETTERE
$B6	LOCAZIONE DI TRASMISSIONE DEL BYTE
$F7-$F8	PUNTATORI DI RICEZIONE DEL BUFFER
$F9-$FA	PUNTATORI DI TRASMISSIONE DEL BUFFER
$0298	N. DEI BITS IN ENTRATA/USCITA
$0299-029A	TEMPO DI TRASMISSIONE DI UN BIT
$029B	INDICE DEL BYTE DELL' ULTIMO CARATTERE RIC.
$029C	INDICE DEL BYTE DEL PRIMO CARATTERE RIC.
$029D	INDICE DEL BYTE DEL PRIMO CARATTERE TRASM.
$029E	INDICE DEL BYTE DELL' ULTIMO CARATTERE TRASM.

ROUTINES DELL' RS-232

$EFA3	ROUTINE DI INGRESSO DELL' NMI
$EFBF	CALCOLO DI PARITA'
$EFE8	CONTATORE DI STOP DEI BITS
$EFEE	ENTRATA ALL' INIZIO DEL BYTE DI TRASM.
$EFFB	FISSA LA RICEZIONE DEL PROSSIMO BYTE
$F016	FISSA L' ERRORE
$F027	CALCOLA IL N DI BITS DA RICEVERE
$F036	ROUTINE DI NMI (COLLECT)
$F040	CALCOLO DI PARITA'
$F046	SHIFT DEI DATI
$F04B	DA LO STOP AI BIT IMMAGAZZINATI NEL BUF.
$F05B	ABILITA' LA RICEZIONE DI UN BYTE
$F068	CONTROLLO DEL BIT DI START
$F06F	METTE I DATA NEL BUFFER
$F08B	CONTROLLO DI PARITA'
$F094	CONTROLLA LA PARITA' CALCOLATA
$F09F	RIPORTO ERRORI

$F0BC	OUTPUT DI UN FILE SU USER PORT
$F0C4	CONTROLLO PER DSR E RTS
$F0CD	CONTROLLO PER ACTIVE INPUT
$F0D4	ATTESA PER CTS BASSO
$F0D9	ATTIVA RTS
$F0E1	ATTESA PER CTS ALTO
$F0ED	MANIPOL. DEL BUFFER PER USCITA CARATTERE
$F0FC	FISSA L' USCITA
$F102	FA PASSARE UN PRIMO BYTE
$F10E	FA PASSARE IL T1 DEL NMI
$F116	INPUT DI UN FILE SU USER PORT
$F122	CONTROLLO TRA DSR E RTS
$F12B	ATTESA PER ACTIVE OUTPUT
$F130	DISATTIVA RTS
$F138	ATTESA PER DCD ALTO
$F13F	ABILITA IL CB1 PER INGRESSO RS-232
$F14F	INPUT DI UN CARATTERE SUL BUFFER
$F15C	RICEVITORE GIRI
$F160	PROTEGGE L' USCITA SERIALE E LA CASSETTA DA UN NMI [

RS-232

Normalmente una porta IEEE-488 e' una porta parallela.
Tuttavia nel VIC questa e' stata implementata come porta seriale. Infatti mentre di norma il BUS DATI e' trasferito parallelamente su una sequenza appunto di 8 linee di dati , sul VIC la trasmissione avviene con 1 linea di dati seriali.
Il bus IEEE del Vic e' costituito da sei linee, tre di input e tre di output. Le tre linee di input controllano le linee di impulso da una periferica al VIC, le tre linee di output hanno una stessa funzione e controllano l'invio di dati dal VIC ad una periferica.
Queste tre linee sono formate da una linea dei dati SERIALI da una linea di CLOCK per il controllo degli impulsi di Clock sulla linea dati e da una linea di ATTENTION.
La funzione della porta seriale IEEE sul VIC e' approssimativamente confrontabile con la stessa uscita implementata sul PET ma e' modificata per numerose applicazioni che consentono l'interfacciamento e la comunicazione tra un VIC e qualsiasi altro tipo di periferiche che possono essere anche altri VIC o computer piu' grandi.
Nel caso venga richiesta una completa IEEE-488 standard sara' necessario usare il COMMODORE IEEE-488 EXPANSION MODULE . Questo consentira' di connettere il VIC a tutte le periferiche che usino la IEEE-488 ed in particolare alle esistenti periferiche del VIC.
Una porta IEEE-488 sia nella semplice versione seriale presente sul VIC, sia nella piena implementazione sul modulo di espansione , presenta notevoli vantaggi rispetto ad una uscita seriale RS 232 o ad un'altro tipo di uscita parallela.
Il vantaggio e' costituito dal fatto che una IEEE-488 ha la capacita' di collegarsi a piu' periferiche attraverso un unico gruppo di linee di I/O .
Ci sono tre tipi di perifeririche che possono

essere collegate al BUS IEEE e sono le seguenti :

Controller - Una unita' che controlla le operazioni del BUS
Listener - Una periferica che riceve dati dal BUS
Talker - Una periferica che trasmette dati sul BUS

Con l'attuale sistema operativo implementato sul VIC solo il VIC puo' agire come un CONTROLLER, anche se in caso di collegamento puo' essere fatto agire sia come Listener che come Talker.
Tutte le periferiche possono quindi essere sia in LISTENER che in TALKER, sebbene solo una periferica per volta possa essere messa in posizione di TALKER.
Il CONTROLLER come dice il nome stesso controlla il trasferimento di dati lungo il Bus e determina quale periferica agisca in funzione di TALKER e quale in funzione di LISTENER.

LE CONNESSIONI DELLA PORTA SERIALE IEEE

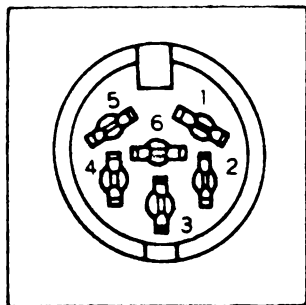
Le sei linee di I/O derivano la loro funzione dai due integrati 6522. La seguente tavola mostra le funzioni ed i collegamenti di ogni linea :

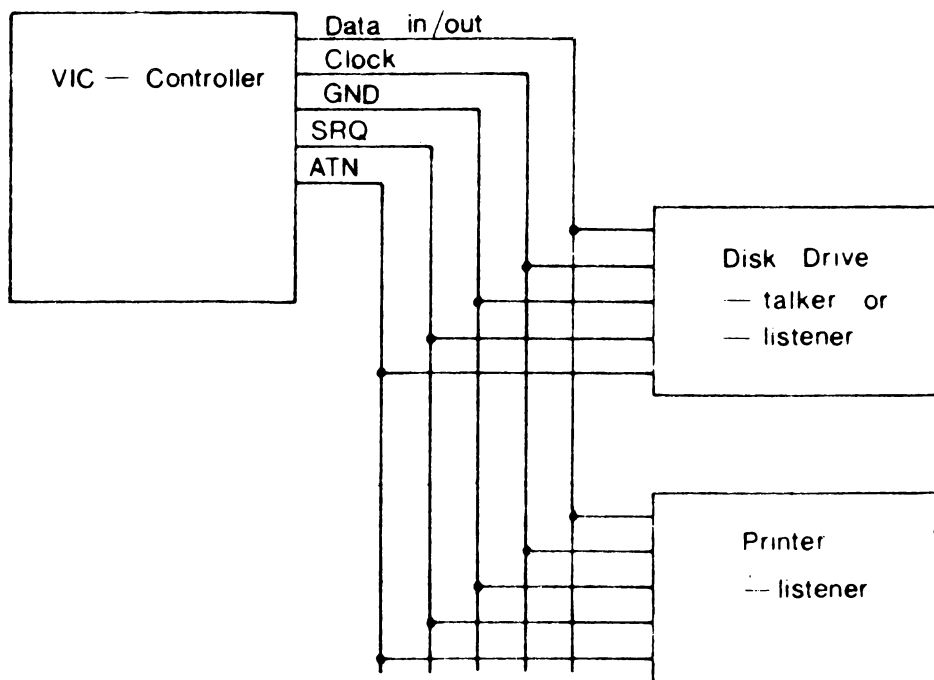
VIA No.	LINEA No.	FUNZIONI DELLA LINEA
VIA1	PA1	SERIAL DATA IN
VIA2	CB2	SERIAL DATA OUT
VIA1	PA0	SERIAL CLOCK IN
VIA2	CA2	SERIAL CLOCK OUT
VIA1	PA7	SERIAL ATN OUT
VIA2	CB1	SERIAL SRQ IN

E' da notare che la linea " ATN IN " e' semplicemente connessa al PIN 9 del connettore USER PORT ma non implementata.
Nel caso venga richiesta la funzione ATN IN allora il PIN 9 dovra' essere collegato ad una linea HANDSHAKE non usata della USER PORT e dovra' essere scritto un appropriato programma per implementare

questa funzione.
Quando venga usata la porta IEEE sul VIC sia nella versione standard che nella espansione esterna con Expansion Module, la sintassi dei comandi e' identica in quanto la differenza e' solo sul modo di trasmettere i dati.

APERTURA DI UN CANALE DELLA IEEE

Comando : OPEN lf,d,sa,"fn"
lf - Normale file logico (1-255)
d - N. della periferica (4-30) Serve per selezionare la periferica che deve ricevere questo gruppo di comandi.
sa - Indirizzo secondario (0-31) Il valore di questo codice e' usato per determinare il modo di operare di una periferica intelligente.
Infatti cambiando l'indirizzo secondario le caratteristiche operative della periferica possono essere cambiate.
fn - Nome del file. Il nome del file e' un'estensione del concetto dell'indirizzo secondario ed e' usato principalmente per le periferiche come nastro o disco.
Il nome del file puo' essere sia una stringa che una variabile fino ad una lunghezza massima di 128 caratteri .
Il comando OPEN seleziona una periferica che puo' avere il valore fra 4 e 30 e dal quel momento il sistema operativo considera quella periferica come una periferica IEEE. Se non e' specificato nessun indirizzo secondario o nessun nome del f ile allora non viene passato niente alla periferica tramite il controller.
Il sistema operativo prende le variabili del comando OPEN e le immagazzina in FILE TABLES . Se invece un nome del file e' dichiarato il sistema perativo manda una sequenza di " LISTEN ATTENTION " alla periferica specificata nel comando di OPEN.
Il sistema operativo del VIC consente che siano aperte fino ad un massimo di 10 file logici per volta.

PIN#	TYPE
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NC

PIEDINATURA DELL'USCITA SERIALE IEEE

CONNESSIONE DELLE PERIFERICHE ALL'USCITA IEEE

RICEVIMENTO DATI DA UN CANALE IEEE

Comando: INPUT£ o GET£ lf,V

lf - file logico come visto nel comando OPEN

V - input di dati immagazzinati nelle variabili V o V$

I caratteri provenienti da una periferica in modo di trasmissione sono accettati tramite il comando INPUT£ ed immagazzinati nella variabile V.
Questa fase viene eseguita fino a quando non venga ricevuto un cosi' detto carattere di delimitazione costituito in questo caso da un Ritorno Carrello o CHR$(13).
Questo tipo di variabile stringa e' immagazzinato nel BASIC INPUT BUFFER che ha una lunghezza massima di 88 caratteri per cui e' necessario che la stringa non superi questa lungheza prima del CHR$(13).
Usando invece il comando GET£, poiche' non viene usato il BUFFER anzidetto non esiste la limitazione degli 88 caratteri.
Tutti e due i comandi di INPUT£ e di GET£ eseguono la stessa sequenza.
Per primo viene chiamata la routine di inizializzazione della IEEE che invia un "TALK ATTENTION" alle periferiche seguito dall' indirizzo secondario specificato per quel File nel comando OPEN.
Alla fine della sequenza di ATTENTION il VIC si posiziona come LISTENER ed attende un segnale dalla periferica chiamata che indica di essere pronta a ricevere un carattere.
Se il segnale non e' ricevuto entro 64 ms viene generato un errore il cui codice e' messo nella variabile ST. Se invece il segnale e' ricevuto allora si passa alla routine di input da IEEE.
Questa routine riporta un singolo carattere dal BUS

usando la linea di CLOCK.
Se durante la fase di INPUT dei dati viene ricevuto un segnale di EOI allora la routine di IEEE fissera' il flag di status della EOI, cio' stara' ad indicare che il prossimo Byte ricevuto sara' l'ultimo della sequenza.

Dopo questa condizione sara' richiamata la routine di " termination untalk" che restituisce la funzionalita' alla tastiera (o meglio la reinserisce) ed invia un comando appunto di UNTALK al Bus IEEE affinche' questo stesso Bus possa essere reso libero per i prossimi comandi.

TRASMISSIONE DATI AD UN CANALE IEEE

Comando : PRINT£ lf,V

lf - File logico

V - Uscita dati immagazzinati nella variabile V o V$

Il comando di PRINT£ per prima cosa richiama una routine del sistema operativo che invia un'ordine di " LISTEN ATTENTION ' alla periferica richiamata dal Bus. Cio' consente di fissare questa periferica nella funzione di " LISTENER ".
Questa operazione e' seguita dal Byte dell'indirizzo secondario specificato nel file logico del comando OPEN.
Il sistema operativo del VIC attende un segnale di risposta dalla periferica definita come LISTENER per un periodo di 256 ms. e non ricevendolo segnala un errore del tipo " ? DEVICE NO PRESENT ' .
La routine di uscita della IEEE trasmette i dati Bit per Bit . L'uscita dati e' immagazzinata nel Buffer prima della trasmissione ed e' di qui che la routine di uscita attinge ogni Byte.
Quando deve essere trasmesso ogni Byte il sistema operativo invia un segnale EOI al Listener per avvertire la periferica che la trasmissione e'

vicina alla fine.
Dopo aver trasmesso questo ultimo Byte il V1C invia un' ordine di " Unlisten " al Bus e riporta la funzione di uscita allo schermo . Cio' libera il Bus per la successiva operazione.

CHIUSURA DI UN CANALE IEEE

Comando : CLOSE lf

lf - File logico

Quando un file IEEE che era stato aperto viene richiuso e' emessa una sequenza di segnali. Questa sequenza di comandi invia l'indirizzo secondario dell'OPEN sommato a $EO alla periferica specificata .Questo consente che un comando di chiusura del file possa essere indirizzato a periferiche intelligenti.

LOCAZIONI DI MEMORIA PER LA IEEE-488

$90	FLAG DI STATO DI I/O
$94	FLAG DEL CARATTERE BUFFERIZZATO
$95	CARATTERE BUFF. DELL' IEEE
$97	TEMPORARIZZAZIONE PER IEEE
$98	PUNTATORE ALLA TABELLA DEI FILE
$9A	N DELLA PERIFERICA IN INPUT
$9B	OUTPUT CMD DEVICE
$A3	FLAG DEL BIT SERIALE DI CONT/EOI
$A4	CONTATORE DI CICLO PER I/O SERIALE
$B7	LUNGHEZZA DEL NOME DEL FILE
$B8	N. DEL FILE
$B9	INDIRIZZO SECONDARIO
$BA	N. DEL DEVICE
$BB	INDIRIZZO DEL NOME DEL FILE
$0200	BUFFER DI 88 BYTES
$0259	TAVOLA DEI LOGICAL FILE, 10 BYTES
$0263	TAVOLA DEI N. DELLE PERIFERICHE, 10 BYTES
$026D	TAVOLA DEGLI INDIRIZZI SECONDARI, 10BYTES
$0285	FLAG DEL TIMEOUT DELLA IEEE

LOCALIZZAZIONE DELLE SUBROUTINES DELL' IEEE- SERIALE

$E4A0	METTE LA LINEA DATA ALTA
$E4A9	METTA LA LINEA DATA BASSA
$E4B2	DEBOUNCE PIA
$EE14	ORDINE DI COMUNICARE AL SERIAL BUS DEVICE
$EE17	ORDINE DI RICEVERE AL SERIAL BUS DEVICE
$EE40	USCITA DI UN BYTE DAL BUS SERIALE
$EE6F	PREDISPONE PER RICEVERE DATI
$EEC0	INVIA L' INDIRIZZO SECONDARIO DOPO ASCOLT.
$EEC5	RICHIEDE ATTENT. DOPO ASCOLTO
$EECE	RISPONDE IL SECONDO INDIRIZZO
$EED3	RISPONDE ATTENT.
$EEE4	USCITA BUFFER AL BUS SERIALE
$EEF6	MANDA IL COMANDO UNTALK AL BUS SERIALE
$EF04	MANDA IL COMANDO UNLISTEN AL BUS SERIALE
$EF19	RICEVE UN BYTE SUL BUS SERIALE

$EF84	FISSA LA LINEA CLOCK ALTA
$EF8D	FISSA LA LINEA CLOCK BASSA
$EF96	RITARDO DI 1 ms

INDIRIZZI DEI VETTORI DI SALTO

$FF93	INVIA INDIRIZZO SECONDARIO DOPO ASCOLTO
$FF96	INVIA INDIRIZZO SECONDARIO DOPO TALK
$FFA2	FISSA IL TIMEOUT SUL BUS IEEE
$FFA5	RICEVE UN BYTE DALLA IEEE
$FFA8	INVIA UN BYTE SULLA IEEE
$FFAB	ORDINE DI UNTALK AL BUS SERIALE
$FFAE	ORDINE DI UNLISTEN AL BUS SERIALE
$FFB1	ORDINE DI LISTEN AL BUS SERIALE
$FFB4	ORDINE DI TALK AL BUS SERIALE
$FFBA	FISSA IL PRIMO ED IL SEC. INDIRIZZO LOGICO
$FFBD	FISSA IL NOME DEL FILE
$FFC0	ESEGUE IL COMANDO OPEN
$FFC3	ESEGUE IL COMANDO CLOSE

M A P P A   D I   M E M O R I A

==== I N D I R I Z Z I ====		F U N Z I O N E
ESA	DECIMALE	
0000-0002	0-2	funzione USR
0003-0004	3-4	converte floating in intero
0005-0006	5-6	converte intero in floating
0007	7	contatore generale del BASIC. Ricerca il carattere ":" o la fine della linea
0008	8	delimitatore di modo apici,00 come limite
0009	9	posizione del cursore
000a	10	flag di verifica
000b	11	puntatore del buffer di input
000c	12	DIM flag.Primo carattere della matrice
000d	13	Variable flag. ff=stringa 00=numerico
000e	14	Integer flag. 80= integer 00=float. point
000f	15	Flag scansione DATA.LIST Flag.Memory flag
0010	16	Subscript flag.FNx flag
0011	17	Flags per :0=input;64=get;1=read
0012	18	Flag per segno ATN
0013	19	attuale canale di I/
0014-0015	20-21	Indirizzo basic per SYS,GOTO,ecc
0016	22	Stack Pointer per temporary string
0017-0018	23-24	Ultimo vettore per temporary string
0019-0021	25-33	Stack per stringhe temporanee
0022-0025	34-37	area di utilizzo per punt.
0026-002a	38-42	area di utilizzo per molt.
002b-002c	43-44	puntatore inizio basic
002d-002e	45-46	punt.fine programma e inizio variabili
002f-0030	47-48	punt.fine variabili inizio array
0031-0032	49-50	punt.fine array
0033-0034	51-52	punt.inizio zona stringhe
0035-0036	53-54	punt.fine zona stringhe
0037-0038	55-56	punt.fine memoria
0039-003a	57-58	n. della attuale linea Basic
003b-003c	59-60	successiva linea Basic
003d-003e	61-62	puntatore per com.Basic (CONT)
003f-0040	63-64	n. della linea DATA

0041-0042	65-66	punt. per il DATA in esame
0043-0044	67-68	vettore di input
0045-0046	69-70	nome della var. corrente
0047-0048	71-72	indirizzo della var. corrente
0049-004a	73-74	puntatore per FOR-NEXT
004b-004c	75-76	salva Y,salva oper.,salva punt. basic
004d	77	maschera per l'operatore in uso
004e-0053	78-83	ara di utilizzo : punt. alla descrizione della stringa, lunghezza della stringa,etc.
0054-0056	84-86	vettore di salto per funzioni
0057-0060	87-96	area di utilizzo per valori num.
0061-0066	97-102	Accumulatore £1: E,M,M,M,M,S
0068	104	Accumulatore £1: Overflow
0069-006e	105-110	Accumulatore £2: Esponente,etc
006f	111	comparazione segno
0070	112	Accumulatore £1: basso ordine
0071-0072	113-114	lungh. buffer cassetta
0073-008a	115-138	Subrtn: Get;7A,7B = puntatore (CHARGOT)
008b-008f	139-143	area di memorizzazione per RND
0090	144	flag per le operazioni di I/o
0091	145	flag per il tasto di stop
0092	146	temporizzatore
0093	147	flag per LOAD o VERIFY
0094	148	uscita seriale, flag caratt. riman.
0095	149	caratt.bufferizzato per IEEE
0095	150	sincronizz. cassetta
0097	151	temporizz. per INPUT sulla IEEE
0098	152	n. di files aperti
0099	153	sgn. dellla perif.in INPUT
009a	154	periferica in OUTPUT per CMD
009b	155	controllo di parita' per cass.
009c	156	dipolo per cassetta
009d	157	flag per mess. da S.O. per RUN o per modo dir.
009e	158	tempor. cassetta. Errore 1
009f	159	" " Errore 2
00a0-00a2	160-162	Jiffy clock
00a3	163	Contatore seriale per bit
00a4	164	Cont. di ciclo per I/O seriale
00a5	165	cont. per scrittura nastro

00a6	166	puntatore per Buffer di cass.
00a7-00b6	167-182	area di lavoro per RS-232
00b7	183	numero dei carat. nel nome del file
00b8	184	numero del file logico corrente
00b9	185	indirizzo secondario corrente
00ba	186	numero della periferica corrente
00bb-00bc	187-188	puntatore al nome del file
00be	190	£ resto del blocco per R/W
00bf	191	buffer della parola seriale
00c0	192	controllo switch cassetta
00c1-00c2	193-194	indirizzo di start per cass.
00c3-00c4	195-196	puntatore kernal
00c5	197	valore del tasto premuto
00c6	198	numero dei caratteri nel buffer
00c7	199	flag di reverse (0 off)
00c8	200	puntatore di fine linea per input
00c9-00ca	201-202	posiz. cursore (riga e colonna)
00cb	203	stesso del 197
00cc	204	lampeggio cursore (0=on 1=off)
00cd	205	flashing cursore
00ce	206	carattere prima del cursore
00cf	207	flag lampeggio del cursore
00d0	208	input da schermo e da tastiera
00d1-00d2	209-210	puntatore alla linea dello schermo
00d3	211	posizione del cursore nella linea
00d4	212	flag delle " (0=off 1=on)
00d5	213	lunghezza delle linee sullo schermo
00d6	214	riga dello sch. in cui si trova il cur.
00d7	215	valore ascii dell'ultimo tasto prem.
00d8	216	flag in modo INSERT
00d9-00f1	217-241	tavola delle linee dello schermo
00f2	242	riga segnata sullo,schermo
00f3-00f4	243-244	puntatore del colore dello schermo
00f5-00f6	245-246	puntatore tastiera
00f7-00f8	247-248	RS-232 puntatore di ricezione
00f9-00fa	249-250	RS-232 puntatore di trasmissione
00fb-00ff	251-255	loc. in pag. zero delle rout. kernal
0100-010a	256-266	area di lavoro ascii
010b-013e	267-318	area di errore nastro
013f-01ff	319-511	area stack del processore
0200-0258	512-600	buffer della riga basic

0259-0262	601-610	tavola del file logico
0263-026c	611-620	tavola del num. della perif.
026d-0276	621-630	tavola dell'indirizzo secondario
0277-0280	631-640	buffer della tastiera
0281-0282	641-642	punt. dell'inizio della memoria
0283-0284	643-644	punt. della fine memoria
0285	645	flag del fuori tempo per IEEE
0286	646	codice del colore corrente
0287	647	codice del colore prima del cursore
0288	648	loc.di base dello schermo (MSB)
0289	649	massima dimensione del buffer tastiera
028a	650	flag di repeat(0=solo cursore, 128=tutti)
028b	651	ritardo prima del repeat
028c	652	ritardo tra i repeat
028d	653	flag shift/control tastiera (shift=1, commodore = 2 , ctrl = 4)
028f-0290	655-656	punt. decod. della tastiera
0291	657	loc. modo shift
0292	658	auto scroll basso (0=on 0=off)
0293	659	RS-232 registro di controllo
0294	660	RS-232 registro di comando
0295-0296	661-662	non standard
0297	663	RS-232 registro di stato
0298	664	numero dei bit da inviare
0299-029a	665-666	baud rate (full) bit time
029b	667	RS-232 punt. di ricez.
029c	668	RS-232 punt. di input
029d	669	RS-232 punt. di trasm.
029e	670	RS-232 punt. di outp.
029f-02a0	671-672	mantiene IRQ durante oper. su nastro
02a1-02ff	673-767	liberi
0300-0301	768-769	vettore di salto per errore
0302-0313	770-787	vettori di salto per basic
0314-033b	788-827	vettori di salto rout. kernal
033c-03fb	828-1019	buffer della cassetta
0400-0fff	1024-4095	area di espansione da 3 K
1000-1dff	4096-7679	area RAM residente
1e00-1fff	7680-8191	area RAM per memoria di schermo in configurazione base
2000-3fff	8192-16383	area per espans. da 8K (blocco 1)
4000-5fff	16384-24575	area per espans. da 8K (blocco 2)

6000-7fff	24576-32767	area per espans. da 8K (blocco 3)
8000-8fff	32768-36863	generatore di caratteri (4 K)
9000-93ff	36864-37887	blocco 0 di I/O
9000	36864	bit da 0 a 6 , centratura orizz.
9001	36865	centratura verticale
9002	36866	bit da 0 a 6 , numero delle colonne bit 7 fa parte dell'ind. per la matr. video
9003	36867	bit da 1 a 6 , numero delle righe bit 0 sceglie i caratt. 888 o 1688
9004	36868	
9005	36869	bit da 0 a 3 , start per gener. di caratt. bit da 4 a 7 , indir. per matrice video

bit 3 2 1 0		ind. di start per gen. carat. esa	decimale
0 0 0 0	rom	8000	32768
0 0 0 1		8400	33792
0 0 1 0		8800	34816
0 0 1 1		8c00	35840
1 0 0 0	ram	0000	0000
1 0 0 1		xxxx	xxxx
1 0 1 0		xxxx	xxxx
1 0 1 1		xxxx	xxxx
1 1 0 0		1000	4096
1 1 0 1		1400	5120
1 1 1 0		1800	6144
1 1 1 1		1c00	7168

9006	36870	posizione orizzontale penna ottica
9007	36871	posizione verticale penna ottica
9008	36872	valore della paddle X
9009	36873	valore della paddle Y
900a	36874	frequenza oscillatore 1 (bassa) da 128 a 255
900b	36875	frequenza oscillatore 2 (media) da 128 a 255
900c	36876	frequenza oscillatore 3 (alta)

		da 128 a 255
900d	36877	frequenza generatore di rumore bianco
900e	36878	bit da 0 a 3 , volume del suono
		bit da 4 a 7 , colore ausiliario
900f	36879	registro del video e colore del bordo
		bit da 4 a 7 , colore del sottofondo
		bit da 0 a 2 , colore del bordo
		bit 3 , selezione modo normale o inverso
9110-911f	37136-37151	6522 VIA 1
9110	37136	registro di uscita porta B
		(user port e rs232)

PIN	DESCRIZIONE	ABBR.
C-PB0	Received data	Sin
D-PB1	Request to Send	RTS
E-PB2	Data termina ready	DTR
F-PB3	Ring indicator	Ri
H-PB4	Received line signal	DCD
J-PB5	Unassigned	XXX
K-PB6	Clear to send	CTS
L-PB7	Data set ready	DSR
B-CB1	Interrupt for Sin	Sin
M-CB2	Transitted data	Sout
A-GND	Protective ground	GND
N-GND	Signal ground	GND

9111	37137	registro di uscita porta A

PIN	DESCRIZIONE
PA0	Serial CLK IN
PA1	Serial DATA IN
PA2	Joystcks 0
PA3	Joystcks 1
PA4	Joystcks 2
PA5	Fire button e Light pen
PA6	Cassette switch sense
PA7	Serial ATN out

9112	37138	registro B di direzione dati
9113	37139	registro A di direzione dati
9114	37140	Timer 1 (byte basso)
9115	37141	Timer 1 (byte alto e start)
9116	37142	Timer 1 (byte basso)
9117	37143	Timer 1 (byte alto)
9118	37144	Timer 2 (byte basso)
9119	37145	Timer 2 (byte alto)
911a	37146	Registro di shift
911b	37147	Registro di controllo ausiliario
911c	37148	Registro di controllo periferico (CA1, CA2, CB1, CB2)
		CA1 tasto Restore
		CA2 Controllo motore cassette
		CB1 Interrupt for Sin
		CB2 Transmitted data
911d	37149	Registro di Interrupt
911e	37150	Abilitazione registro di Interrupt
911f	37151	registro di uscita porta A (senza handshake)
9120-912f	37152-37167	6522 VIA 2

9120	37152	Registro di uscita porta B (scansione colonne della tastiera)
		PB3 Cassette write line
		PB7 Joysticks 3
9121	37153	Registro di uscita porta A (scansione righe della tastiera)
9122	37154	Registro B di direzione dati
9123	37155	Registro A di direzione dati
9124	37156	Timer 1 (byte basso latch)
9125	37157	Timer 1 (byte alto start)
9126	37158	Timer 1 (byte basso latch)
9127	37159	Timer 1 (byte alto latch)
9128	37160	Timer 2 (byte basso latch)
9129	37161	Timer 2 (byte alto latch)
912a	37162	Registro di shift
912b	37163	Registro di controllo ausiliario
912c	37164	Registro di controllo periferico
		CA1 Cassette read line

		CA2	Serial clock out
		CB1	Serial SRQ in
		CB2	Serial data out
912d	37165		Registro di Interrupt
912e	37166		Abilitazione registro di Interrupt
912f	37167		Registro di uscita porta A (senza handshake)
9400-95ff	37888-38399		RAM del colore con espansione nel blocco 1
9600-97ff	38400-38911		RAM del colore in configurazione base
9800-9bff	38912-39935		blocco 2 di I/O
9c00-9fff	39936-40959		blocco 3 di I/O
a000-bfff	40960-49151		Espansione ROM da 8K
c000-dfff	49152-57343		ROM da 8K contenente il BASIC
e000-ffff	57344-65535		ROM da 8K contenente il Sistema Oper.

ADC
Add memory to accumulator with carry

Somma il contenuto della locazione di memoria specificata con il contenuto dell' Accumulatore sommando anche il bit di Carry.
Il risultato dell' operazione viene riportato nell'Accumulatore.

AND
And memory With accumulator

Esegue ua operazione di amd logico fra il contenuto dell' accumulatore ed il contenuto dellA locazione di memoria specificata.
Il risultato della operazione viene riportato nell' Accumulatore.

ASL
Shift left one byte

Esegue uno spostamento verso sinistra del contenuto dell' Accumulatore o della locazione di meoria specificata.
Il bit piu' significativo del contenuto di partenza viene riportato nel Carry.
Il bit che si libera viene riempito con uno 0.

BCC
Branch on Carry clear

Esegue il salto specificato se ilbit di Carry e' uguale a 0

BCS
Branch on Carry set

Esegue il salto specificato se ilbit di Carry e' uno

BEQ
Branch on result zero

Esegue il salto specificato se il bit di Zero e' uguale a uno

BIT
Test bit in memory whith Accumulator

Esegue un AND logico fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata.
Il risultato dell' Operazione non compare nei registri della macchina, pero' se il risultato e' =0 il bit ZERO dello status viene messo a uno.
In caso contrario si ha Z=0.

BMI
Branch on result minus

Esegue il salto specificato se il bit di negativo e' uguale a uno.

BNE
Branch on result not zero

Esegue il salto specificato se il bit di Zero =0.

BPL
Branch on result plus

Esegue il salto specificato se il bit di Negativo = 0.

BRQ
Force break.

Forza una interruzione non mascherabile tramite il bit I dello Status.
Sava automaticamente il PC e pi. Mette a 1 il bit I dello Status.

BVC
Branch on overflow clear.

Esegue il salto specificato se il bit di Overflow e' = 0.

BVS
Branch on Overflow set

Esegue il salto specificato se il bit di Overflow e' uguale a 1

CLC
Clear Carry flag

Forza a zero il bit di Carrydello status.

CLD
Clear decimal mode

Forza a zero il bit di decimal dello status.
Le operazioni aritmetiche svolte successivamente a questa istruzione sono svolte in binario.

CLI
Clear interrupt disable bit

Forza a zero il bit di interruptdello Status.
Dopo questa istruzione la CPU e' abilitata a servire le interruzioni che dovessero arrivarle.

CLV
Clear Overflow flag

Forza a uno il bit di Overflow dello Status.

CMP
Compare memory and Accumulator

Confronta il contenuto della locazione di memoria specificata con il contenuto dell' Accumulatore.
CPX
Compare memory and index X

Confronta il contenuto della locazione di memoria specificata con il contenuto del Registro X.

CPY
Compare memory and index Y

Confronta il contenuto della locazione di memoria specificata con il contenuto del Registro Y.

DEC
Decrement memory by 1

Toglie una unita' al contenuto della locazine di memoria specificata.

DEX

Decrement index X by 1

Toglie una unita' al contenuto del Registro indice X

DEY
Decrement index Y by 1

Toglie una unita' al contenuto del Registro indice Y.

EOR
Exclusive or memory whith Accumulator

Esegue una operazione di OR esclusivo fra il contenuto della locazione di memoria specificata ed il contenuto dell' Accumulatore.
Il risultato dell' operazione viene riportato nell' Accumulatore.

INC
Increment memory by 1

Somma al contenuto della locazione di memoria specificata 1.

INX
Increment index X by 1

Somma 1 al contatore del Registro indice X

INY
Increment index Y by 1

Somma 1 al contenuto del registro indice Y.

JMP
Jump to new location

Esegue un salto incondizionato alla locazione di memoria specificata con indirizzo assoluto.

JSR
Jump to new location serving return adress

Salta ad un nuovo indirizzo salvando in Stack il PC di partenza

LDA
Load accumulator whith memory

Carica in accumulatore il contenuto della locazione di memoria specificata.

LDX
Load index X with memory

Carica nel registro indice X il contenuto di memoria specificata.

LDY
Load index Y with memory

Carica nel registro indice Y il contenuto della locazione di memoria specificata.

LSR
Shift right one bit(memory or accumulator)

Esegue uno spostamento verso destra del contenuto dell' Accumulatore o della locazione di memoria

specificata.
Il bit meno significativo del contenuto di partenza viene riportato nel Carry.
Il bit che si libera viene riempito con uno 0.

NOP
No operation

Non esegue alcuna operazione significativa.

ORA
Or memory whth accumulator

Esegue un OR logico fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata.
Il risultato della operazione viene riportato nell' Accumulatore

PHA
Push accumulator on stack

Esegue la memorizzazione dell' Accumulatore nello Stack.
Lo Stack Pointer viene decrementato di uno.

PHP
Push processor status on Stack

Esegue la memorizzazione dello Status nello Stack
Lo Stack Pointer viene decrementato di uno
PLA
Pull Accumulator from Stack

Esegue il caricamento dell' Accumulatore con il contenuto della locazione di Stack puntata dallo Stack Pointer

Lo Stack Pointer viene incrementato di uno.

PLP
Pull processor status from Stack

Esegue il caricamento dello Status con il contenuto della locazione puntata dallo Stack pointer.
Lo Stack Pointer viene decrementato di uno

ROL
Rotate one bit left(memory or accumulator)

Esegue la rotazione verso sinistra del contenuto dell' Accumulatore o del contenuto della locazione di memoria specificata.

ROR
Rotate one bit right (memory or accumulator)

Esegue la rotazione verso destra del contenuto dell' Accumulatore o del contenuto della locazione di memoria specificata.

RTI
Reuturn from interrupt

Esegue il ritorno dalla interruzione ripristinando dallo Stack lo status ed PC.

RTS
Return from subroutine

Esegue il ritorno da una subroutine ripristinando dallo Stack il PC.

SBC
Subtract memory from Accumulator whith borrow

Esegue la sottrazione aritmetica fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata, sottraendo anche il Carry negato.
Il risultato della operazione viene riportato nell' Accumulatore.

SEC
Set Carry flag

Forza a uno il bit di Carry dello Status.

SED
Set decimal mode

Forza ad uno il bit di decimal dello Status.
Le operazioni aritmatiche svolte successivamente a questa istruzione sono effettuate in modo decimale.

SEI
Set interrupt disable status

Forza a uno il bit di disabilitazione dell' interruzione dello Status.
Dopo questa distruzione la CPU non risponde piu' alle richieste di interruzione.

STA
Store accumulator in memory

Esegue la scrittura del contenuto dell' Accumulatore nella locazione di memoria specificata.

STY
Store index Y in memory

Esegue la scrittura del contenuto del registro indice Y nella locazione di meoria specificata.

TAX
Transfer Accumulator to index X

Copia il contenuto dell' Accumulatore nel registro indice X.
L' Accumulatore rimane invariato.

TAY
Transfer Accumulator to index Y

Copia il contenuto dell' Accumulatore nel registro indice Y.
L' Accumulatore rimane invariato

TYA
Transfer index Y to Accumulator

Copia il contenuto del registro Y nell' Accumulatore.
Il registro Y rimane invariato

TSX
Transfer Stack Pointer to index X.

Copia il contenuto dello Stack Pointer nel registro indice X
Il registro S rimanane invariato.

TXA
Transfer Index X to Accumulator

Copia il contenuto del Registro indice X nell' Accumulatore.
Il registro X rimane invariato.

TXS
Transfer index X to Stack Pointer

Copia il contenuto del registro indice x nello Stack Pointer
Il registro X rimane invariato

```
1 REM------------------------------
2 REM------------------------------
3 REM-------- DISASSEMBLER --------
4 REM------------------------------
5 REM---------- BY EVM ------------
6 REM------------------------------
7 PRINT"■":GOSUB119
8 DATABRK,ORA,?,?,?,ORA,ASL,?
9 DATAPHP,ORA,ASL,?,?,ORA,ASL,?
10 DATABPL,ORA,?,?,?,ORA,ASL,?
11 DATACLC,ORA,?,?,?,ORA,ASL,?
12 DATAJSR,AND,?,?,BIT,AND,ROL,?
13 DATAPLP,AND,ROL,?,BIT,AND,ROL,?
14 DATABMI,AND,?,?,?,AND,ROL,?
15 DATASEC,AND,?,?,?,AND,ROL,?
16 DATARTI,EOR,?,?,?,EOR,LSR,?
17 DATAPHA,EOR,LSR,?,JMP,EOR,LSR,?
18 DATABVC,EOR,?,?,?,EOR,LSR,?
19 DATACLI,EOR,?,?,?,EOR,LSR,?
20 DATARTS,ADC,?,?,?,ADC,ROR,?
21 DATAPLA,ADC,ROR,?,JMP,ADC,ROR,?
22 DATABVS,ADC,?,?,?,ADC,ROR,?
23 DATASEI,ADC,?,?,?,ADC,ROR,?
24 DATA?,STA,?,?,STY,STA,STX,?
25 DATADEY,?,TXA,?,STY,STA,STX,?
```

```
26 DATABCC,STA,?,?,STY,STA,STX,?
27 DATATYA,STA,TXS,?,?,STA,?,?
28 DATALDY,LDA,LDX,?,LDY,LDA,LDX,?
29 DATATAY,LDA,TAX,?,LDY,LDA,LDX,?
30 DATABCS,LDA,?,?,LDY,LDA,LDX,?
31 DATACLV,LDA,TSX,?,LDY,LDA,LDX,?
32 DATACPY,CMP,?,?,CPY,CMP,DEC,?
33 DATAINY,CMP,DEX,?,CPY,CMP,DEC,?
34 DATABNE,CMP,?,?,?,CMP,DEC,?
35 DATACLD,CMP,?,?,?,CMP,DEC,?
36 DATACPX,SBC,?,?,CPX,SBC,INC,?
37 DATAINX,SBC,NOP,?,CPX,SBC,INC,?
38 DATABEQ,SBC,?,?,?,SBC,INC,?
39 DATASED,SBC,?,?,?,SBC,INC,?
40 DIMT$(255):FORI=0TO255:READT$(I):NEXT
41 INPUT"DEC. ADDRESS";A
42 PRINT"]";
43 FORM=1TO11
44 I=PEEK(A)
45 K=(I)AND15
46 IFI=0THENL=1:GOTO65
47 IFK=8THENL=1:GOTO65
48 IFK=10THENL=1:GOTO65
49 IFK<>0THEN53
50 IFI=32THEN57
```

```
51 IF(I)AND16THEN53
52 IFI<128THENL=1:GOTO65
53 IFK=0THEN61
54 IFK=9THEN58
55 IFK<10THEN64
56 J=PEEK(I)
57 L=3:GOTO65
58 K=(I)AND16
59 IFK=0THEN64
60 GOTO57
61 K=(I)AND240
62 IFK>111THEN64
63 IFK=32THEN57
64 L=2
65 N=A:GOSUB110
66 PRINT"    ";
67 IFT$(I)="?"THENL=1
68 FORX=ATOA+L-1
69 N=PEEK(X):GOSUB110:PRINT"  ";
70 NEXT
71 PRINT:PRINT"■■■■■■■■■■■■■■■■■";
72 IFT$(I)<>"?"THEN74
73 PRINTCHR$(13)"■.BYT   $";:N=I:L=1:GOSUB110:GOTO101
74 PRINTCHR$(13)"■"T$(I);
75 IFL=1THEN101
```

```
76 PRINT"    ";
77 X=(I)AND15:Y=(I)AND240
78 IFL=3THEN92
79 ONX+1GOTO80,86,85,10000,89,89,89,10000,10000,85
80 IF(Y)AND16THEN82
81 GOTO85
82 N=PEEK(A+1):IFN>127THENN=-(256-N)
83 PRINT"$";:N=A+N+2
84 GOSUB110:GOTO101
85 PRINT"#$";:N=PEEK(A+1):GOSUB110:GOTO101
86 PRINT"($";:N=PEEK(A+1):GOSUB110
87 IF(Y)AND16THENPRINT"),Y";:GOTO101
88 PRINT",X)";:GOTO101
89 PRINT"$";:N=PEEK(A+1):GOSUB110
90 IF(Y)AND16THENPRINT",X";
91 GOTO101
92 ONX+1GOTO95,8,8,8,8,8,8,8,8,93,8,8,98,94,94
93 PRINT"$";:N=PEEK(A+2):GOSUB110:N=PEEK(A+1):GOSUB110:PRINT",Y";:GOTO101
94 IFI=200THEN93
95 PRINT"$";:N=PEEK(A+2):GOSUB110:N=PEEK(A+1):GOSUB110
96 IF(Y)AND16THENPRINT",X";
97 GOTO101
98 IFI=198THEN95
99 IFI<>108THEN95
100 PRINT"($";:N=PEEK(A+2):GOSUB110:N=PEEK(A+1):GOSUB110 :PRINT")";
```

```
101 PRINT:PRINT"[illegible]:";
102 FORJ=1TOL
103 X=PEEK(A+J-1)
104 IFX<32THENX=X+32
105 IFX>127THENX=X-128:GOTO104
106 PRINTCHR$(X);:NEXT
107 A=A+L
108 PRINT:   NEXT
109 GOTO41
110 A$="0123456789ABCDEF"
111 H$=""
112 K=N-INT(N/16)*16
113 N=INT(N/16)
114 H$=MID$(A$,K+1,1)+H$
115 IFN>0THEN112
116 IFLEN(H$)=1THENPRINT"0";
117 PRINTH$;
118 RETURN
119 FORI=1TO2000
120 PRINT"[illegible]DISASSEMBLATORE[illegible]":
121 RETURN
```

```
1 REM ************************************
2 REM *                                  *
3 REM *           P L O T                *
4 REM *                                  *
5 REM ************************************
6 PRINT"]"
7 POKE36869,253
8 POKE36866,143
9 POKE36867,162
10 POKE56,20
11 POKE55,0
12 POKE650,128
13 FORI=832TO869:READA:POKEI,A:NEXT:SYS832
14 DATA160,6,162,255,202,138,157,,30,152,157,,150,224,,208,243
15 DATA160,8,169,,162,,157,,20,202,208,250,238,89,3,136,208,242,96,,
16 POKE36879,30
17 XX=X:YY=Y:GETA$
18 IFA$="L"THENX=X+1:GOSUB27:GOTO17
19 IFA$="J"THENX=X-1:GOSUB27:GOTO17
20 IFA$="M"THENY=Y+1:GOSUB27:GOTO17
21 IFA$="I"THENY=Y-1:GOSUB27:GOTO17
22 IFA$="L"THENX=X+1:GOSUB33:GOTO17
23 IFA$="\"THENX=X-1:GOSUB33:GOTO17
24 IFA$="\"THENY=Y+1:GOSUB33:GOTO17
25 IFA$=","THENY=Y-1:GOSUB33:GOTO17
```

```
26 GOTO17
27 IFX<0ORX>119THENX=XX:RETURN
28 IFY<0ORY>135THENY=YY:RETURN
29 A%=X/8:B%=Y/8:R1%=X-A%*8:R2%=Y-B%*8:V%=2↑(7-R1%)
30 C%=5120+B%*120+A%*8+R2%:VV%=PEEK(C%):CC%=C%
31 POKEC%,V%ORVV%
32 RETURN
33 POKECC%,PEEK(CC%)-V%
34 IFX<0ORX>119THENX=XX:RETURN
35 IFY<0ORY>135THENY=YY:RETURN
36 A%=X/8:B%=Y/8:R1%=X-A%*8:R2%=Y-B%*8:V%=2↑(7-R1%)
37 C%=5120+B%*120+A%*8+R2%:VV%=PEEK(C%):CC%=C%
38 POKEC%,VV%ORV%
39 RETURN
```

```
1 REM ------------------------------------
2 REM ------------------------------------
3 REM ---------PROGRAMMA  MUSIC----------
4 REM ---------       BY          ----------
5 REM ---------  J. S. BACH     ----------
6 REM ------------------------------------
7 REM ------------------------------------
8 PRINT"       J.S. BACH"
9 PRINT"INVENZIONE IN FA MAGG."
10 PRINT:PRINT
11 PRINT"SEI ALL'OPERA PUOISCEGLIERE LA VELOCITA'DI ESECUZIONE
12 PRINT"SI CONSIGLIA DI":PRINT"INIZIARE CON VELOCITA'100"
13 PRINT:PRINT:PRINT:PRINT
14 DIMA(16)
15 POKE36878,12
16 REM VELOCITA'
17 INPUT"VELOCITA'";Q
18 FORI=0TO16:READA(I):NEXT
19 READX:IFX=-1THENRESTORE:GOTO17
20 POKE36876,A(X)
21 READY
22 POKE36875,A(Y)
23 FORJ=1TOQ:NEXT
24 GOTO19
25 DATA0,195,201,207,209,215,219,221,225,228,231,232,235
```

```
26 DATA237,238,240,223
27 DATA4,0,0,0,6,0,0,0
28 DATA4,0,0,0,8,0,0,0
29 DATA4,0,0,0
30 DATA11,0,11,0,10,4,9,0
31 DATA8,6,9,0,8,4,7,0
32 DATA6,8,7,0,6,4,5,0
33 DATA4,11,0,11,6,10,0,9
34 DATA8,8,0,9,6,8,0,7
35 DATA11,6,0,7,8,6,0,5
36 DATA13,4,15,0,14,6,15,0
37 DATA13,8,15,0,14,6,15,0
38 DATA13,11,15,0,14,8,15,0
39 DATA11,13,13,15,12,14,13,15
40 DATA11,13,13,15,12,14,13,15
41 DATA11,13,13,15,12,14,13,15
42 DATA9,11,11,13,10,12,11,13
43 DATA9,11,11,13,10,12,11,13
44 DATA9,11,11,13,10,12,11,13
45 DATA16,9,0,11,5,10,0,11
46 DATA9,9,0,11,16,10,0,11
47 DATA11,9,0,11,9,10,0,11
48 DATA12,16,13,0,12,5,11,0
49 DATA10,8,11,0,10,5,9,0
50 DATA8,10,9,0,8,8,7,0
```

```
51 DATA6,11,0,12,9,11,8,10
52 DATA16,9,8,10,16,9,6,8
53 DATA5,16,6,8,5,16,4,6
54 DATA3,5,4,0,3,8,2,16
55 DATA1,6,0,16,8,6,16,5
56 DATA8,4,0,5,3,4,0,3
57 DATA4,2,0,3,8,2,0,1
58 DATA3,5,0,4,8,3,0,4
59 DATA2,5,0,0,16,5,0,0
60 DATA8,1,8,1,8,1,0,0,-1
```

ISTRUZIONI PER IL PROGRAMMA CHARACTER SET

Alla sinistra del video appaiono i comandi di cui esaminiamo le funzioni:

Disp = serve per selezionare quale carattere della INFORMATION LINE posta in fondo allo schermo si vuole visualizzare nella matrice. Alla domanda quindi rispondere con un numero da 0 a 63.

Goto = serve per passare ad un' altro carattere sempre da 0 a 63.

Load = carica la matrice di un carattere preventivamente salvato su nastro o su disco. Ricordarsi che si hanno a disposizione 512 Bytes a partire dalla locazione decimale 7168.

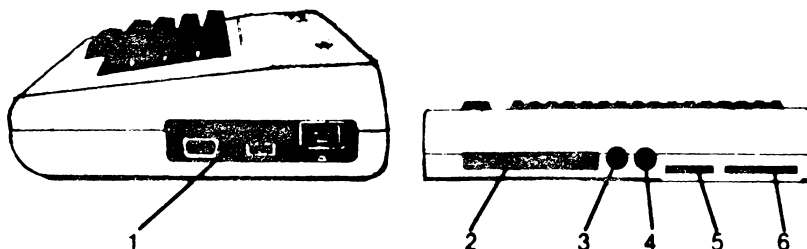
Mask = richiede appuntoun MASK per un particolare carattere. Il programma chiedera' il SET (0=upper case/graphics, 1=upper/lower case) ed il numero del carattere (da 0 a 127).Il VIC cambiera' il carattere attuale con il nuovo e dara' anche la possibilita' di modificarlo.

Next = passa al successivo carattere della sequenza.

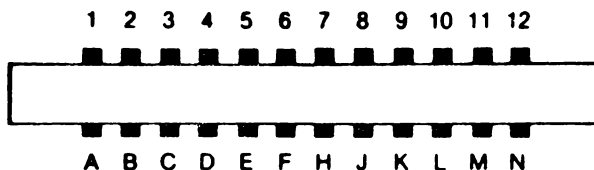
Save = carica su nastro o su disco i nuovi 64 caratteri creati. Ovviamente in questo set possono essere presenti anche vecchi caratteri che avremo copiato dalla ROM del generatore.

Repl = simile al comando MASK solo che invece di prelevare il carattere dal generatore ROM lo carica dalla parte RAM cioe' da quelli creati.

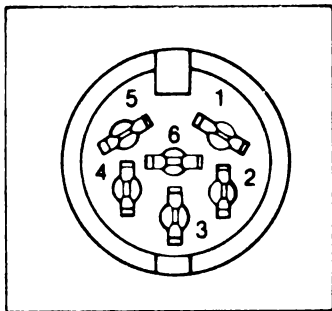
```
1 PRINT"CHARACTER EDITORBY ANDY FINKEL(C)1981 BY COMMODORE
2 POKE51,0:POKE52,28:POKE55,0:POKE56,28:CL=30720:C0=32768:C2=7168:SB=7680:B=7841
:B1=8120:
3 DEFFNA(I)=N*8+I+Y:DEFFNB(J)=-((ZAND2↑(7-J))>0)*170-((ZAND2↑(7-J))=0)*160
4 V=36869:B2=7873:BL$="                    ":PRINTCHR$(8):POKE650,128:IFPEEK(V)=2
55THEN10
5 POKEV,255:PRINT"KEEP CHARS?";:GOSUB59:IFA$<>"N"THEN10
6 PRINT"MASK CHARS?";:GOSUB59:IFA$="N"THENFORI=0TO511:POKEI+C2,0:NEXT:GOTO10
7 PRINT"SET(0:1)";:GOSUB59:I=VAL(B$)
8 PRINT"BLOCK(0:3)";:GOSUB59:A=VAL(B$):IFA<0ORA>3ORI<0ORI>1THEN7
9 A=A*64+I*256:FORI=0TO511:POKEI+C2,PEEK(I+C0+A*8):NEXT
10 PRINT"DISP H,W?";:GOSUB59:DL=VAL(LEFT$(B$,1))-1:DW=VAL(RIGHT$(B$,1))-1
11 IFDW<0ORDL<0ORDW>2ORDL>3THEN10
12 RESTORE:PRINT"":A=1:N=0:FORI=0TODL:FORJ=0TODW:POKEB2+I*22+J,0:NEXTJ,I
13 READJ,K:IFJ>0THENFORI=0TO7:POKEJ*8+SB+I,PEEK(K*8+32768+I):NEXT:GOTO13
14 DATA6,160,7,170,8,131,9,136,10,129,12,146,13,148,19,133,30,147,35,132,41,137,
42,143,0,0
15 PRINT"";:FORI=8098TO8119:POKECL+I,3:POKEI,173:NEXT
16 FORI=B1TOB1+63:POKEI+CL,6:POKEI,I-8120:NEXT:GOSUB52
17 PRINT"CMDSDISPGOTOLOADMASKNEXTQUITREPLSA
VE
18 CP=B:X=0:Y=0
19 CS=PEEK(CP):POKECP,-(CS=160)*70-(CS=170)*71
20 GETA$:IFA$=""THEN20
21 IFA$="L"THENPRINT"NAME ";:GOSUB59:POKEV,240:LOADB$,1,1:CLR:RUN
22 IFA$=""THENX=0:Y=0:GOTO51
23 IFA$=""THENFORI=0TO7:POKEC2+N*8+I,256+NOT(PEEK(C2+N*8+I)):NEXT:GOSUB52:GOTO1
8
24 IFA$=""THENFORI=0TO7:POKEN*8+C2+I,0:NEXT:GOSUB52:GOTO18
```

```
47 IFA$="■"THENX=X+1+(X>6)
48 IFA$="■"THENX=X-1-(X<1)
49 IFA$="■"THENY=Y-1-(Y<1)
50 IFA$="■"THENY=Y+1+(Y>6)
51 CS=PEEK(CR):POKECR,-(CS=70)*160-(CS=71)*170:CR=B+X+Y*22:GOTO19
52 PRINT"■■■■■■"SPC(7)MID$(BL$,3,9)"■■■■■■"SPC(7)"■■CHR #"NCHR$(13)SPC(7)"■---
-----■"
53 FORI=0TO7:Z=PEEK(C2+N*8+I):FORJ=0TO7:POKEB+I*22+J,FNB(J):POKEB+I*22+J+CL,6:NE
XTJ,I
54 POKEB1+N+CL,7:POKESB+362+CL,6:POKESB+362,N:RETURN
55 PRINT"■■■SET#:CHR#?";:GOSUB59:A=VAL(LEFT$(B$,1))*256+VAL(MID$(B$,3)):IFA>511
ORA<0THEN55
56 GOTO58
57 PRINT"■■■CHAR#?";:GOSUB59:A=VAL(B$):IFA<0ORA>63THENPRINTBL$:GOTO57
58 PRINTBL$:RETURN
59 B$=""
60 GETA$:IFA$=""THEN60
61 IFASC(A$)=13THENPRINTBL$:A$=LEFT$(B$,1):RETURN
62 IFASC(A$)=20ANDLEN(B$)=0THEN60
63 IFASC(A$)=20THENB$=LEFT$(B$,LEN(B$)-1):PRINT"■ ■";:GOTO60
64 IFLEN(B$)>4ORASC(A$)<31THEN60
65 B$=B$+A$:PRINT"■"A$;:GOTO60
```

1) Game I/O
2) Memory Expansion
3) Audio and Video
4) Serial I/O
5) Cassette
6) User Port

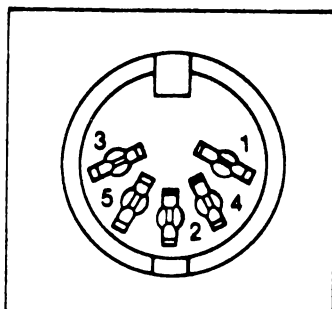
DISPOSIZIONE DEI VARI CONNETTORI

USER I/O

PIN #	TYPE	NOTE	PIN #	TYPE	NOTE
1	GND		A	GND	
2	+5V	100mA MAX.	B	CB1	
3	$\overline{\text{RESET}}$		C	PB0	
4	JOY0		D	PB1	
5	JOY1		E	PB2	
6	JOY2		F	PB3	
7	LIGHT PEN		H	PB4	
8	CASSETTE SWITCH		J	PB5	
9	SERIAL ATN IN		K	PB6	
10	+9V	100mA MAX.	L	PB7	
11	GND		M	CB2	
12	GND		N	GND	

PIEDINATURA DELL'USER PORT

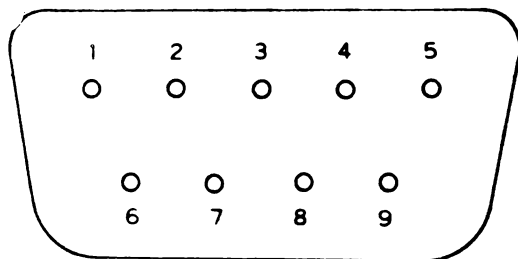
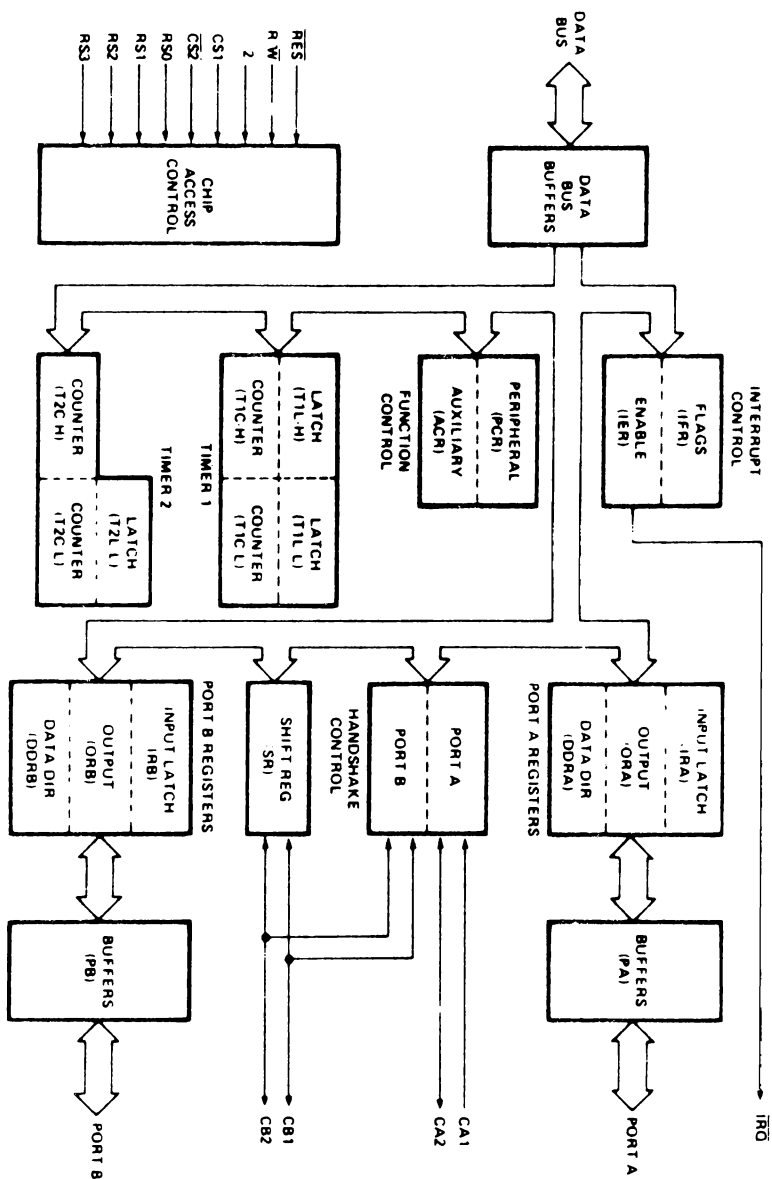
SERIAL I/O

PIN #	TYPE
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NC

PIEDINATURA DEL CONNETTORE DELL'USCITA SER.

AUDIO/VIDEO

PIN #	TYPE	NOTE
1	+6V	10mA MAX
2	GND	
3	AUDIO	
4	VIDEO LOW	
5	VIDEO HIGH	

PIEDINATURA DELL'USCITA AUDIO/VIDEO

GAME I/O

PIEDINATURA DELL'USCITA GIOCHI

PIN #	TYPE	NOTE
1	JOYØ	
2	JOY1	
3	JOY2	
4	JOY3	
5	POT Y	
6	LIGHT PEN	
7	+5V	MAX. 100mA
8	GND	
9	POT X	

STRUTTURA INTERNA DEL 6522

Register Number	RS Coding				Register Desig.	Description	
	RS3	RS2	RS1	RS0		Write	Read
0	0	0	0	0	ORB/IRB	Output Register "B"	Input Register "B"
1	0	0	0	1	ORA/IRA	Output Register "A"	Input Register "A"
2	0	0	1	0	DDRB	Data Direction Register "B"	
3	0	0	1	1	DDRA	Data Direction Register "A"	
4	0	1	0	0	T1C-L	T1 Low-Order Latches	T1 Low-Order Counter
5	0	1	0	1	T1C-H	T1 High-Order Counter	
6	0	1	1	0	T1L-L	T1 Low-Order Latches	
7	0	1	1	1	T1L-H	T1 High-Order Latches	
8	1	0	0	0	T2C-L	T2 Low-Order Latches	T2 Low-Order Counter
9	1	0	0	1	T2C-H	T2 High-Order Counter	
10	1	0	1	0	SR	Shift Register	
11	1	0	1	1	ACR	Auxiliary Control Register	
12	1	1	0	0	PCR	Peripheral Control Register	
13	1	1	0	1	IFR	Interrupt Flag Register	
14	1	1	1	0	IER	Interrupt Enable Register	
15	1	1	1	1	ORA/IRA	Same as Reg 1 Except No "Handshake"	

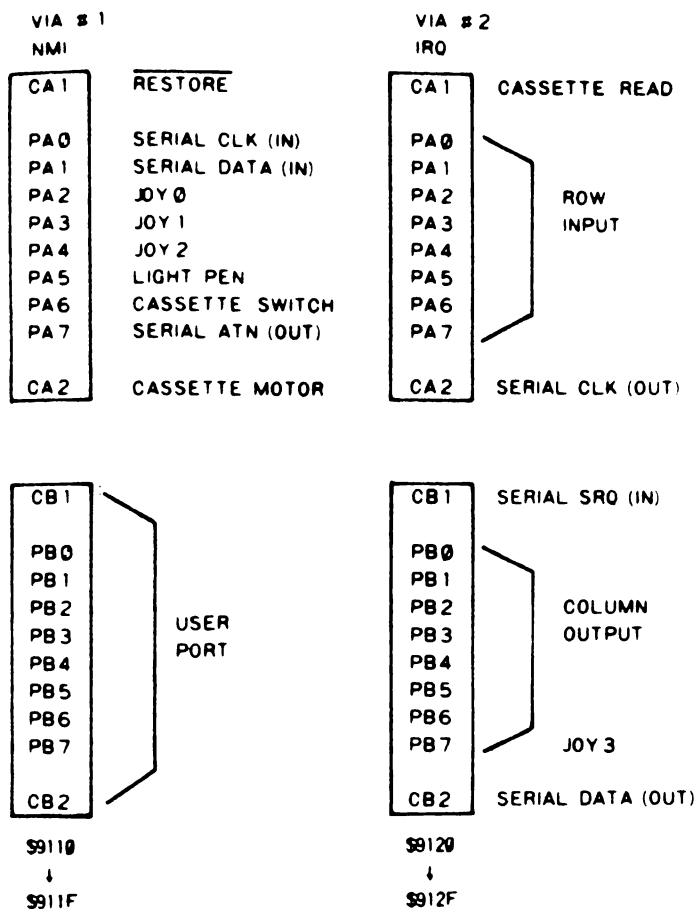
REGISTRI DEL 6522

ASSEGNAZIONE DELLE LINEE DI I/O
DEI DUE 6522

TAVOLA DI CONVERSIONE ESADECIMALE/DECIMALE

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0		0		0		0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	E	3,840	F	240	F	15

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

TAVOLA DI CONVERSIONE ESADECIMALE/DECIMALE

BIT 7	BIT 6	BIT 5	OPERATIONS
		0	Parity disabled, none Generated/Received
0	0	1	Odd Parity Receiver/Transmitter
0	1	1	Even Parity Receiver/Transmitter
1	0	1	Mark Transmitted Parity Check Disabled
1	1	1	Space Transmitted Parity Check Disabled

REGISTRO DI COMANDO RS-232

SCHEMA ELETTRICO

CIRCUITO 1

CIRCUITO 1

CIRCUITO 2

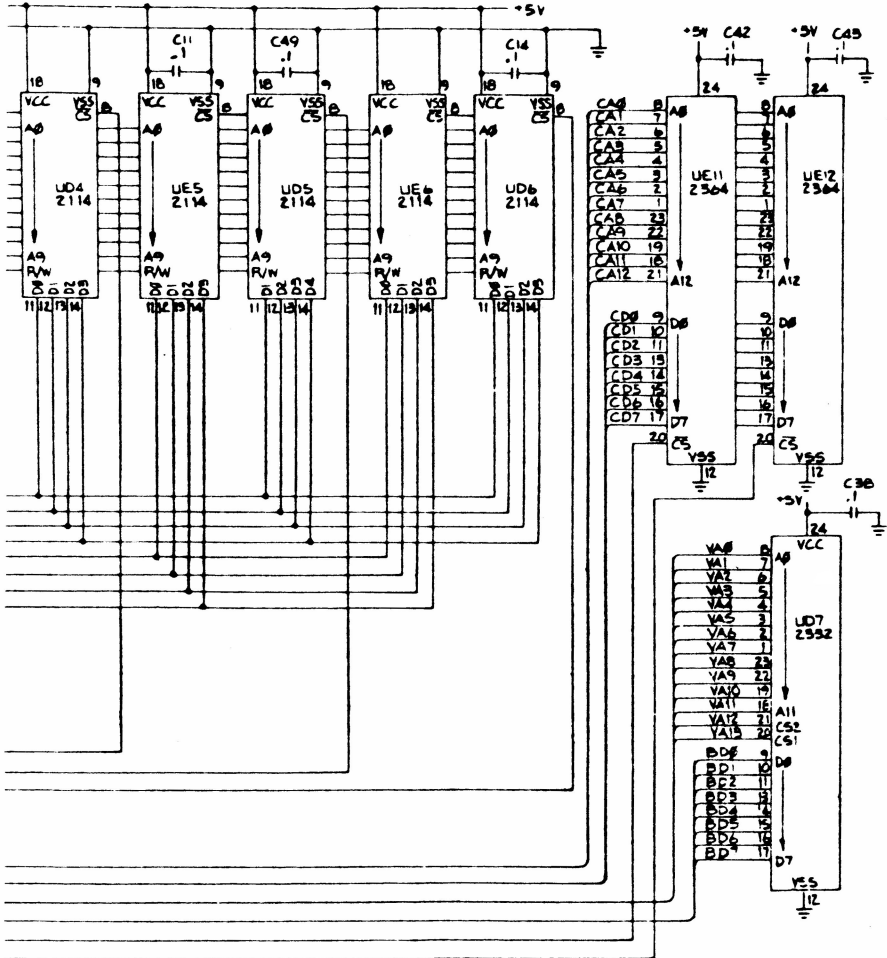
RES
BDØ-BD7
CAØ-CA15
CR/W
SØ2
+5V
C1
.1
UAB1
6522
VCC
IRQ
CA1
CA2
DØ
D7
PAØ
PA7
PBØ
PB7
RSØ
RS3
CS1
CS2
R/W
Ø2
RES
CB1
CB2
VSS
ROW Ø
ROW 1
ROW 2
ROW 3
ROW 4
ROW 5
ROW 6
ROW 7
COL Ø
COL 1
COL 2
COL 3
COL 4
COL 5
COL 6
COL 7
JOY 3
SERIAL CLK OUT
RESTORE
KEY
KEYBOARD CONN
SERIAL DATA OUT
UB4
7406
FB4
FB5
FB3
R5
1K
R6
1K
R4
1K
SERIAL SRQ IN
SERIAL PORT P3
R10
1K
UAB3
6522
R1
10K
C5
CASS MOTOR
SERIAL CLK IN
SERIAL DATA IN
JOY Ø
JOY 1
JOY 2
LITE PEN
CASS SWITCH
SERIAL ATN OUT
SERIAL ATN IN
I/OØ
USER PORT P1
CNT
AC9V

CIRCUITO 2

CIRCUITO 3

CIRCUITO 3

+5V
C11
.1
C49
.1
C14
.1
C42
.1
C43
.1
C38
.1
VCC
VSS
CS
A0
A9
R/W
UD4
2114
UE5
2114
UD5
2114
UE6
2114
UD6
2114
UE11
2364
UE12
2364
UD7
2332
A12
D0
D7
A11
CS2
CS1

GUIDA AL PERSONAL VIC 20 COPIA N. ________

Segnalazione di errore alla pagina .....

Riferentesi a..........................

....

....

....

....

SUGGERIMENTI...........................

....

....

....

....

Nome e cognome.........................

Via............................N....

Città............................( )

Sono in possesso di....................

Con le seguenti periferiche............

....

....

SPEDIRE IN BUSTA A:

E.V.M Computers
Via Marconi 9/a
52025 MONTEVARCHI (AR)